AN INTELLIGENT FORM SYSTEM

by

Heyun Liu, B.Eng.

A thesis submitted for the
requirements of the Doctor of Philosophy

School of Computing and Management Sciences
Sheffield Hallam University
Hallamshire Business Park
100 Napier Street
Sheffield S11 8HD
United Kingdom

August, 1992

Abstract

This thesis presents an investigation of developing a user-centred formbase system. It is based on the previous developments in Office Information Systems. It is technically related to AI Planning Systems and Database Systems.

An Office System is an open system inside which the data, as well as the operations upon the data can not be pre-defined exactly. In order to set up a stable and flexible information system in such environments, the task representation, activity representation and data representation must be dynamically related to each other. This research concerns how to use AI planning system concepts to develop a formbase system. There are three crucial aspects: (a) how to represent an activity of information processing, (b) how to represent and refer to the data in the forms, and (c) how to construct the problem solving process for the task of information processing. For reasons of flexibility and stability in the open environment, it is important that a proper link between data representation and activity representation is achieved.

This research has generated an Intelligent Form System. The contributions are: (a) the development of a form pattern language and the formbase which can represent and refer to the forms, (b) the identification of the formbase activity schema which can represent the activity upon the forms, and (c) the development of a problem solving process for the information processing tasks of the forms. The research has also recognized that the information processing activities upon forms are very different from the activities which are automatically performed by the Humans.

Key Words: Office Information Systems; AI Planning Systems; Activity Representation; Action Reasoning; Knowledge Representation; Office Form Systems.

Acknowledgements

I would like to thank my supervisors Professor Ian Draffan and Professor Frank Poole for their help and advice during the research. Without their constant support, this research would not have been possible.

Dr. Jawed Siddiqi, Dr. Innes Jelly and Dr. Jon Gray have also given help during the project.

I would like to extend my sincerely thanks to the school secretary Marilyn Wilson, the technicians in the school, especially to Mick Fitzgibbons, John Leach and Chris Mather, and to my colleagues in Room HA147 for their warmth and humour.

Finally I would like to thank my parents and my sister for their continued love and consistent encouragement.

List of Figures

Contents

Chapter 1.

Introduction

This thesis presents research which is in the area of Office Information Systems. Technically, it relates to Artificial Intelligent Planning Systems, Knowledge Representation, and Database Systems. Practically, it lays down a foundation for developing form-oriented software.

The aim of the research is to improve the flexibility and stability of an Information System inside which data as well as the procedures that manipulate the data can not be pre-defined. The key issue of the aim is to identify a mechanism so that dynamic task requirements can be fulfilled based on partially well defined procedural knowledge and the situation. This research is based on the earlier developments in Office Information System such as OBE (Office-By-Example) [Zloof, M., 1982], SCOOP [Zisman, 1978], ICN [Ellis, C., 1979], OMEGA [Barber, 1983], POISE [Croft & Lefkowitz, 1984], POLYMER [Croft & Lefkowitz, 1988], OFM [Tsichritzis, D. C., 1982], SOS [Bracchi, 1984] and OPAS [Lum, V. Y., 1982].

1.1 Rationale of Research

The development of an Office Information System is driven by the requirements of functionality, flexibility and stability. These requirements can be understood in three aspects: the data representation, the activity representation, and the link between the activity representation and the data representation.

The data representation should be independent from the functions that the system supports, and insensible to the modification operations. It should also be able to represent integrity rules upon the data [Gibbs, 1985].

The activity representation should provide a flexible and stable procedural knowledge representation which can adapt to the changes in the dynamic environments and support the tasks that users require. This means that the modification operations upon procedural knowledge representations should not influence the functionality of the system, and a new functional requirement should not result in big changes of the structures of procedural knowledge. In other words, the activity representation should achieve certain independence from the functions that the system provides.

For the link between office data and office activities, there are two aspects: On one hand the activity representation should be able to access the data they need in the database, on the other hand knowledge of the state that an activity representation needs should be provided by the information in the database. The first aspect reflects the information processing features of the system, the second aspect reflects the constraints that the background situation of an office activity has for the information processing process.

If we review the history, for a simple data processing system, the functions that the system should support are simple and can be formally defined. There is no difference between tasks and activities. The requirements for flexibility and stability are mainly at the database level. The cash point services that are provided by banks are a typical example. The procedures that a cash point should support are simple and limited - withdraw cash, check balance, list statements, and so on. Since there are no temporal relationships or data communications between these procedures/operations/tasks/activities, the structure that is required to support them are simple and straightforward. Problem solving processes are not involved. The stability and flexibility of the systems are required only for the modification operations of the database system.

For an Office Information System, the situation is totally different. The data as well as the functions that the system should support can not be formally pre-defined. If we insisted on using traditional methods to model it, we can do nothing more than identifying a large group of independent processes and then supporting them. This is what the first generation of office system models have done. OFFICETALK-ZERO [Ellis, C., 1979], for example, is an integration of several sub-systems and only supports disconnected simple operations. OBE [Zloof, 1982] is also in this category, the operations that OBE can support are not more than database manipulations. This is far away from our expectations for an Office Automation System, since its functionality is too simple.

To improve the functionality of an office model, it is necessary to model the functional and temporal interrelationships between office procedures/activities. SCOOP [Zisman, 1978] and ICN [Ellis, C., 1979] were early systems which modelled office procedures. The structures that they used to support office functionality were no longer straightforward, instead they used networks to represent the system functions. A systems task is decomposed into sub-procedures which are represented as nodes inside the network. This enables SCOOP and ICN to represent the temporal the relationships as well as data communications between the sub-procedures. However, the mechanism that both SCOOP and ICN used for supporting office task is not semantically based, therefore the structure of the system is not stable. This is because a node in the network is defined by identifying a set of operations which can be connected with others through a distinct state condition. This identifying process is more like a simple division of a continuous procedure rather than a construction of a network from the semantically distinct nodes. Thus the network structure is related to the functions that the system supports. Therefore, the system only can support the task requirements which are expected. Moreover, since both of them have taken every minor detail of office procedures and office data, even minor changes in office procedures and office data may cause big alteration of the system structure, which is difficult to carry on since semantic reasons are not used as the basis for system construction.

Therefore, the structures proposed by SCOOP and ICN have to be abandoned, office procedures must be represented in a form that possesses more semantic meanings. The strongest semantic factor in an office system is the office task. So office procedural knowledge representation should be augmented by their goals [Barber, 1984]. In this way procedural knowledge can be organised by a goal which the procedure wants to achieve. Since goals are semantically more distinct, the knowledge encapsulated within them is more stable. Moreover, after the augmentation of goals with an office procedure representation, the functionality of an office information system can be supported by a problem solving process dynamically. Therefore, the flexibility and functionality of the system are determined by the ability of the problem solver. This is obviously a natural structure for any intelligent system. OMEGA [Barber, 1983], POISE [Croft & Lefkowitz, 1984] and POLYMER [Croft & Lefkowitz, 1989] are efforts toward this direction in OIS modellingresearch. They initially discussed the problems of supporting an office information system task, such as how to represent and how to coherently decompose a task.

However, an office task is not so easy to support as the task of a simple data processing system. If an office task involves personal problem solving efforts and a large quantity of communications, it will unavoidably fall into the problems of formalizing social conception of knowledge and action [Gasser, L., 1991]. The developed models, such as OMEGA and POLYMER, were not able to approach these problems. In fact, their approaches to activity representation and problem solving are not connected with data representation. Therefore they have an innate difficulty of developing cognitive problem solving processes since human problem solving strategies are usually embedded in a situation [Suchman, 1990]. Thus the three aspects of an office information system: the data representation, the activity representation and the link between them, should be considered together. Task decomposition and subtask cooperation are not the whole story of modellingan office system, office data modellingmust be coherent with activity modellingand task supporting.

There is a category of developed office models which deals with these three aspects together. They are the form-based office models which include FFM [Tsichritzis, D. C., 1982], OPAS [Lum, V.Y., 1982] and SOS [Bracchi and Pernic, 1984]. The most important feature that a form-based office model has is that it can use a form pattern to define not only the data inside data base, but also the conditions and the effects of an activity and the requirements of a task. Therefore, there is a natural coupling between the data representation, activity representation and task representation. This is essential for a system whose environment is open and dynamic. However, the developed form-based office models have limitations both in their representational ability and in their problem solving ability. On the representational side, either data representation, activity representation or task representation are restricted by the form patterns that the system used. The form patterns are basically relational, for example the reference pattern used by FFM [Tsichritzis, D. C., 1982], and can not represent the office forms which mostly are nested and can not be normalized. On the problem solving side, developed form-based office models have not explored a cognitive approach, instead most of the systems explore procedural representation with central concern of data modeling. Therefore, researches in this area tend to be the same as Object Oriented Database Systems.

In this thesis, an Intelligent Form System (IFS) is presented, it includes a Formbase System which can model office forms in an organization and a problem solver, called Formbase Activity Problem Solver (FAPS), which can assist the problem solving for the activities upon the forms. Office forms and the information processing activities upon them are usually consciously designed by an organization to perceive and to control the performance of the organization. The research that is presented in this thesis has identified a nice combination of data modellingand problem solving which is able to cut the social 'corner' for identifying a cognitive problem solving process. Since every organization deals with a large amount of forms, a computer aided form processing system has very large application potential. In the Formbase System, the formbase query, formbase predicates and formbase operations are developed based on FFM [Tsichritzis, D. C., 1982] and SOS (Semantic Office System) [Bracchi, 1984]. The FAPS (Formbase Activity Problem Solver) is developed based on the earlier developments such as OMEGA [Barber, 1983] and POLYMER [Croft & Lefkowitz, 1988], and the AI planning systems, such as NOAH [Sacerdoti, 1975], NONLIN [Tate, 1977], and TWEAK [Chapman, 1987].

1.2 The Outline of the Thesis

Chapter 2 gives a review for the developed office models, then the developed Intelligent Form System (IFS) is presented. The Formbase System of the IFS is presented in Chapter 3 and Chapter 4. Chapter 3 introduces the data definition and data manipulation languages of the Formbase System, Chapter 4 introduces the internal representation and the implementation issue of the Formbase System.

The Formbase Activity Problem Solver (FAPS) is introduced in Chapter 5 and Chapter 6. Chapter 5 present the activity representation mechanism for the activities upon the forms, while Chapter 6 discusses the problem solver. A brief review of the developed AI activity representation mechanism and their corresponding problem solvers can be found in Chapter 5. The conclusions of the research are made in Chapter 7.

Chapter 2.

Background of Research

Many contributions have been made by the developed office models for modellingoffice data, office activities and the link between office data and office activities. The examples of the developed office models include Officetalk-Zero [Ellis, 1979], OBE (Office-By-Example) [Zloof, M., 1982], OFFIS [Konsyniski, et al, 1982], SCOOP [Zisman, 1978], ICN [Ellis, C., 1979], OMEGA [Barber, 1983], POISE [Croft & Lefkowitz, 1984], POLYMER [Croft & Lefkowitz, 1988], OFM [Tsichritzis, D. C., 1982], SOS [Bracchi, 1984] and OPAS [Lum, V. Y., 1982]. This chapter reviews the developed office models in terms of the above three aspects.

2.1 Office Data Modeling

The aim of office data modellingis to form a viewpoint of the objects manipulated by the office workers and support the necessary operations for a viewpoint. At the earlier stage, office data modellingwas the major research issue for office system modeling. There was a group of office models which were developed based on office data modeling. This group of models includes OFFICETALK-ZERO, introduced in [Ellis, C., 1979], OBE (Office-By-Example) [Zloof, M., 1982], and OFFIS [Konsyniski, et al, 1982]. They have initially tackled the information objects that an office model should include, and the necessary operations upon these information objects that an office model should support. Their research results for office data modellingform the basis for the later developments.

Officetalk-Zero, introduced in [Ellis, C., 1979], was a prototype of a first generation office information system. It integrated several subsystems including a text editor, graphic package, communication facilities and filing facilities. All these subsystems are provided via a simple, uniform form interface. The information processing operations are not more advanced than database system manipulations. More operational features of this model can be found in [Ellis, C., 1979].

OBE (Office-By-Example) [Zloof, 1982] is a further development of the data base language QBE (Query-By-Example) [Zloof, 1977]. The difference is that it incorporates more data types such as letters, forms, reports, charts, and graphs. The operations that OBE can perform on these data types are:

a) Cross-reference between fields by entering identical example elements on two or more fields of the same or different objects.
b) Formulating conditions on field values.
c) Moving data from one field to another.
d) Deriving new values
e) Locating text by using partially underlined strings of characters.
f) Distributing objects to a dynamic list of destinations.
g) OBE can express trigger conditions for some simple actions, such as insert, delete, etc.

After the development of data abstraction mechanisms and rule based production systems, office system data modellingis further developed.

Based on these technologies, Gibbs [Gibbs, 1985] identified the requirements for modellingoffice information objects. The requirements are:

a) Using an object to model entities: Objects are used to model individual entities, documents, and even syntax-directed editors and interactive editors/formatters.

b) Abstraction mechanisms: This includes classification, generalization and aggregation.

c) Integrity constraints: This includes template constraints for the values of the attributes of objects; constraints for modification operations upon objects; and semantic constraints for the links between objects.

d) Supporting unformatted data types.

If not considering the requirements for unformatted data types, the data modellingmechanism which was used in SOS (Semantic Office Systems) [Bracchi,1984] can almost match the above requirements. SOS used three data abstraction mechanisms: generalization, aggregation, and association. It also has a rule production system which can access the facts defined in the data model, and therefore it is able to specify integrity rules upon data objects. For example, an office document can be defined by SOS as follow:

```
<document>
 {aggregation-of
   {
     name;
     creator;
     owner;
     ownership;
   }
 }
```

A semantic constraint can be specified as the following [Bracchi, 1984]:

```
<SR1>
{
 is-a static-rule;
 on new program;
 if not (for-each paper part-of program
  such-that exist paper)
 then program-irregularity
 }
```

2.2 Office Process Modeling

Office procedure modellingis the most important part of office modellingresearches. It analyzes and describes office work by looking at different activities performed concurrent by the user and the system. There have been two styles of modellingmethods. One tries to support the routine office procedures by using well defined nets, the other uses problem solving methods developed in AI to support office work.

2.2.1 Well-Defined Network Representation

SCOOP (System for Computerization Of Office Processing) [Zisman, 1978] is the most typical system which uses a well defined network to model office procedures. It is based on Petri Nets [Peterson, 1977] but augmented by a production system. It consists of three parts: a set of rules constituting a condition and an action sequence, a data base allowing data of the state to be maintained, and a rule interpreter. The production system functions by testing the condition of each rule. If the condition is deemed true the consequent actions are performed. The most important contribution that was made by SCOOP system is the discovery of using an internal network representation and a production system to model the office procedures. An important disadvantage of the system is that its data base can not represent the office documents which are manipulated by the office processes.

To improve SCOOP's inability of referring to the office documents that an office process needs, another office information model was proposed. That was ICN (Information Control Net) [Ellis, C., 1979]. ICN defines an office as a set of related procedures. Each procedure consists of a set of activities connected by temporal orderings called "precedence constraints". In order for an activity to be accomplished, it may need information from repositories, such as files or forms. An Information Control Net can capture the notions of procedures, activities, precedence, and repositories, that is, knowledge of (1) the particular data items transferred to or from repositories, (2) who performs the activity, (3) the amount of data transferred by an activity can be attached to it. For more details see reference [Ellis, C., 1979].

Both SCOOP and ICN use well-defined networks to represent procedures, therefore a task of the system has to be an expected request. In another words, the mapping between the task representation and the activity representation can not adapt itself to the open and dynamic environments. Just as Barber pointed out in his paper [Barber, 1983]: "Precisely because of its succinctness a procedural description suffers from two defects: first, it glosses over minor details that may be problematic or critical in practice; second, the reasons for the actions specified by a procedural description must be inferred". Therefore "even routine tasks in offices encounter unexpected obstacles." This is because "in a procedural approach, it is necessary to foresee the possible alternative courses of action when a procedural step cannot be performed." But "determining what the alternatives are is part of what office work is; all alternatives cannot be determined in advance." Thus "As a result, a procedural approach is not a very useful style of work description because it needs to be augmented by the procedure's goal structure. When a procedure is augmented in this way, one can examine the procedure's goal structure in order to generate alternative steps when a step cannot be performed."

2.2.2 Goal-Augmented Activity Representation

To augment a procedure's goal structure to describe a procedure is to find the meta knowledge for developing a procedure net. The process of developing a procedure net is referred to as the problem solving process in the literature of AI planning systems. When a task request is a problem for the problem-solver rather than a simple request to perform a pre-defined routine, the capacity of the systems to support office tasks is dependent on the ability of their problem solving processes. Barber proposed a model in 1983 called OMEGA [Barber, 1983], which had developed a problem solving process to support the problem solving of office work. Croft and Lefkowitz later developed POISE [Croft & Lefkowitz, 1984] and POLYMER [Croft & Lefkowitz, 1988].

OMEGA [Barber, 1983] used a viewpoint reasoning system to support the problem solving of office work. The problem solver of OMEGA was developed based on the developments of the earlier knowledge representing systems, such as QA4, Conniver, FRL and KRL. A shortcoming shared by these earlier developed systems was that they could not reason contradictions because "they are based on logics where truth is a global characteristic of a statement." [Barber, 1983] The improvements made by

OMEGA limited the effects of contradictions within viewpoints. It is similar to ATMS (Assumption Truth Maintaining System) [de. Kleer, J., 1984; 1986].

POISE [Croft & Lefkowitz, 1984] supports a simple mechanism for task decomposition. It consists of three components, they are a procedure library which contains the procedural descriptions; a semantic database which contains descriptions of the objects used in the procedures and descriptions of the available tools; a model of a particular user's state which includes partial instantiations of procedure descriptions with parameters derived from specific user actions, as well as instantiations of semantic database objects. For example, in the procedure library there could be a procedure for filling out a purchase order form. In the semantic database there would be a description of this form, its fields, and its relationships to other forms and fields used in the system. After a user had started to fill in a particular purchase order form, the user model would contain a partial instantiation of the "Fill-out-purchase-order-form" procedure with values derived from the actual values filled in by the user. There would be also an instantiation of the semantic database object that represents the purchase order form.

To represent office procedures, POISE uses a specific language, an example of which is given in the following. [Croft & Lefkowitz, 1984]

```
PROC        Purchase-Items
DESC         (Procedure for purchasing items with nonstate funds.)
IS          (Receive-purchase-request
              '(Process-purchase-order | Process-purchase-requisition)
             '(Complete-purchase))
WITH        ((Purchaser = Receive-pur-request.Form.Purchaser)
             (Items = Receive-pur-request.Form.Items)
              (Vendor-name = Receive-pur-request.Form.Vendor-name))
COND          (for-value {Purchaser Items Vendor-name}
               (eq  Receive-pur-request.Form
                    Process-pur-order.Form
```

```
                          Process-pur-requisition.Form
                          Complete-pur.Form))
PRECONDITIONS-
SATISFACTION      (for-values {purchaser Items Vendor}
                           (exist Complete-pur.Form))
```

The algorithm syntax of the procedure is specified by the IS clause, refined by the COND clause, and its parameters are defined by the WITH clause. The conditions required for a procedure to begin are specified by the PRECONDITION clause, while the goals satisfied by a procedure are contained in the SATISFATION clause.

The IS clause of the procedure definition provides a precise way of describing the standard algorithm for accomplishing a task in terms of other procedures and primitive operations. The sequence of constituent procedures is specified using the operators catenation ('), alternation (I), shuffle (#), optional ({}), plus (+), and star (*). The concatenation operator specifies the exact temporal ordering of two procedures. If only one of two procedures is to occur, the alternation operator is used. Shuffle permits the interleaving of the components of two procedures in any order. The optional operator is used to specify that a procedure may or may not occur. Plus operators allow procedures to occur one or more times, while the star operator, the closure of plus, indicates zero or more occurrences.

The above example is a "Purchase-items" procedure. The IS clause specifies that after a purchase request has been received, either a purchase requisition or a purchase order is processed. The task is completed by the steps in the Complete-purchase procedure.

Obviously, POISE has achieved to some extent the link between office data and office procedures, and can synthesize procedures using a set of operators and procedural primitives. However variables can not be used in an activity representation, the activity representation therefore is not an abstractive template. The users of a POISE system must be familiar with the system details so that they can modify the procedure library (primitive activities) according to the changes in semantic database (office information objects base), and define new office procedures based on defined primitives in the procedure library.

Moreover, events, sub-activities and their relations must be clearly specified. This actually means this kind of representation also can not ignore the minor details of office procedures. Furthermore the activity representation has not encapsulated enough information for office activities. For example, there is no state protection information, therefore the interactions between different activities or office tasks can only be resolved, if possible, by the assistance of integrity rules, and this is with the condition that the designer knows previously that the interaction will happen and how to resolve it.

POLYMER [Croft & Lefkowitz, 1989] intended to further develop the POISE system. It uses the concepts of AI Planning systems, such as the Task Formalisms [Tate, A,. 1984], to represent knowledge of office activities. Thus, an office activity schema in POLYMER contains fields for describing the goal of the activity, steps for fulfilling the activity (decomposition), the temporal and causal relations among the activity's steps, preconditions and side effects of the activity, the agents responsible for performing the activity, and any additional constraints on the activity. An example of an office activity schema of POLYMER system is shown as the follow: [Croft & Lefkowitz, 1988]

```
ACTIVITY: Accept-or-Reject.Way1
          Goal: refereed(?paper)
 Preconditions: member (?paper, papers)
Decomposition: GOAL desion-reached = exists (status (?paper))
        Control: repeat decision-reached until
                 or(status (?paper, "accepted"), status (?paper, "rejected"))
        Agents: ?editor = member (?editor, editors)
```

This activity description reflects the potential cyclic nature of journal editing. It attempts to achieve the goal of reaching a decision on the paper and indicating that this decision is either "accepted" or "rejected". If the decision is anything else, the task will continue. The previous developed systems, such as OMEGA, do not have a mechanism which can acquire procedural knowledge as easily as this office activity schema. Compared with POLYMER, OMEGA also has difficulties in acquiring information objects in an office system, since the reasoning mechanism inside it does not have structures to distinguish knowledge of the situation and knowledge of the activity.

POLYMER aims to develop a goal achieving process. When an office task is input to the system, based on the given activity schemas and the office situation descriptions, the goal achieving process will construct a procedure network for fulfilling the office task dynamically. Using this modellingstyle, the procedural networks are all constructed dynamically by the goal achieving process of the planner. Minor details can be ignored.

However, the problem solving procedure inside POLYMER is problematic. Basically the connection between its activity representation and data representation is loose, the semantic base of its problem solving process is unclear.

In technical details, it uses the networks of the Assumption-based Truth Maintaining System (ATMS) [de. Kleer, J., 1984; 1986] to represent the nodes and the worlds of the system. In this way, "the parents of a world must be specified when a world is created. They may not be modified later, nor may existing worlds be specified as children of a world." Therefore it is not a strong problem solver since it can not properly process interactions.

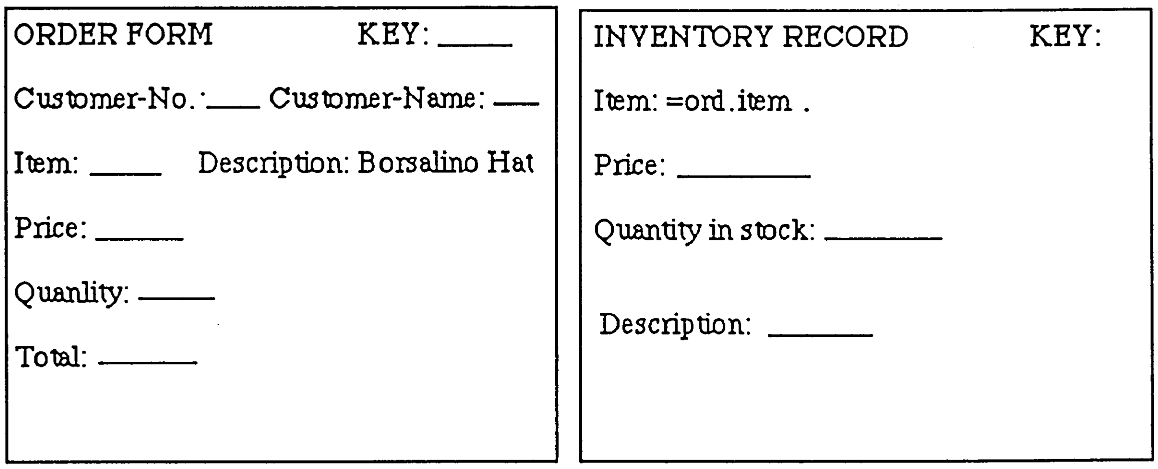
2.3 Link between Data Representation and Activity Representation

The connection between data and activities is one of the essential aspects for modellingan office system. Among the developed office models, form based office models, such as FFM (Form Flow Model) [Tsichritzis, D. C., 1982], SOS [Bracchi, 1984] and OPAS [Lum, V. Y., 1982], have developed a mechanism which can couple data representation and activity representation together. The reason that a systematic approach for modellingoffice work can be developed based on the form concept is because "an office form is a common office information object, it not only provides a structure for organizing office data, but also functions as an interface for office workers to access and to manipulate the data." "The computerized forms are not only the conceptual image of business paper forms, but are more general and elaborate so as to be able to represent any structural data and their templates." [Tsichritzis, D. C., 1982]

The FFM (Form Flow Model) [Tsichritzis, D.C., 1982] views an office as "a network of stations through which forms flow." Information is gathered on 'forms' as structured data, and is processed at one or several stations. 'Station' is the term used for an abstract entity which relates a person, or their role with a physical location and device through which they can operate. A form begins at a station within the network, is processed as it moves through other stations, and ultimately ends at a station within the network. The coordinating of the route which the forms take from station to station is accomplished by the network.

A form type inside FFM has a set of attributes and a set operation procedures. The operations include inserting and modifying a value of an attribute into a form, coping and deleting an instance of a form type, and the operations which deal with moving forms between stations in the system. When forms are flowing within the network, operation requests on form instances are issued from stations.

An office activity inside the FFM model is specified by a system called TLA (Toronto Latest Acronym). A TLA activity is a collection of "sketches." A sketch resembles a form, but is distinguished from form types or form instances. A form precondition sketch is a request to find "a form that looks like this." An action sketch is a request to modify a form that has already been obtained by the precondition sketch. Both of them are called form sketches. The medium of a form sketch specification is the same form template of the form instance being described. Form sketches are used to capture the restrictions referring to values that appear in the forms in the working set locally or globally. Local restrictions are constant attribute values, sets, or ranges of values, and relations between values of the attributes on a given form. Global restrictions on the working set of an automatic procedure are the join conditions between values of attributes appearing on different forms. For example, Figure 2.1(a) is a precondition sketch which instructs the system locally to watch for order forms which are requesting 'Borsalino Hats', and Figure 2.1(b) is a precondition sketch that instructs the system globally to watch for an order of an inventory form. The linking conditions can appear in either sketch.



ORDER FORM KEY: ____

Customer-No.: ____ Customer-Name: ____

Item: ____ Description: Borsalino Hat

Price: ____

Quanlity: ____

Total: ____

INVENTORY RECORD KEY:

Item: =ord.item .

Price: ____

Quantity in stock: ____

Description: ____

(a) (b)

Figure 2.1

Compared to the mechanism of representing activity in POLYMER system or in ICN system, this method of representing an activity could properly couple the data and the activity together, that is, the activity specification has reference patterns for the data of specific forms. Since the form patterns are supported by an interpretation process which is called when necessary, the link between the data and the activity can be dynamic and need not be defined as fixed connections. This makes the activity representation of the system more flexible than models like ICN. On the other hand, since the definition of the activities contain the form patterns, the connection between the data and the activity is not as uncertain as POLYMER.

The problematic aspect of FFM is its approach to mapping a task requirement to activity representations. The strategy that the FFM has used is to find a path among the stations, the stations on the path should issue all the necessary operations for fulfilling the task requirements. This is really an issue of distributed problem solving. As it is shown in Figure 2.2, every station needs to decide what it can do for the form and where to direct it. Therefore, the effects of every station must be recorded, and there must be information for deciding where to send the form next, the algebraic approach of FFM can not deal with the semantic complexity.

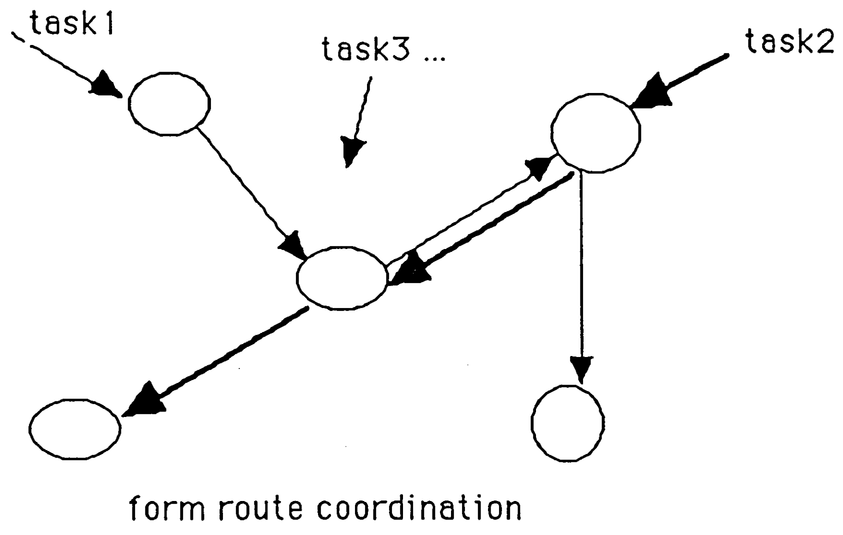


Figure 2.2

SOS (Semantic Office System) [Bracchi, 1984] model is composed of three submodels: static submodel, dynamic submodel and evolutive submodel. The structure is shown in table 1. The static office data is modelled in the Static Submodel. Office processes are supported by the Dynamic and Evolutive Submodels. The integrity constraints are supported by the integrity rules in both static and dynamic submodel. The adaptability of the system is supported in the evolutive submodel by the system evolutive rules.

```
                System Structure of SOS
---------------------------------------------------------
STATIC SUBMODLE
        Office Objects:
            documents
            dossiers
            agents
        Static rules

DYNAMIC SUBMODEL
        Office Activities
        Dynamic rules
              semantic rules
              authorization rules

EVOLUTIVE SUBMODEL
        Office evolutive rules
        System evolutive rules
-----------------------------------------------------
```

table 1

The static submodel is the data model of the office system. It includes static office elements and static rules which are the static integrity constrains. Static office elements are documents, dossiers and agents. The most important contribution of SOS model is the identification of an abstraction mechanism for modellingoffice forms/documents. The mechanism includes generalisation, aggregation and association. An example is shown in the following:

```
<document>
  { aggregation_of
   { name: string;
     creator:agent;
     owner:agent;
      ownership_type:{personal,official};
   }
  }
```

Office activities and tasks are supported in SOS by rules in both the dynamic and evolutive submodels. All the rules in SOS have the following syntax structure. Every rule contains three parts: declaration, condition and the action body.

```
<rule>
{  { declaration:  /* involved elements are listed here */
    { rule-id:        /* rule identifier number */
     is-a...;
       involved-elements
      aggregation-of: ...}} /* end of declaration */
  {condition: /* for invocation or termination */
   { aggregation-of
     {temporal-condition:...
      on-event-condition:...
      on-element-condition:if...}}} /* end of condition */
 { body:then body; }
}  /* end of rule */
```

An example of activity can be specified as follow:

```
<selecting-paper> is-a activity
 { aggregation-of
  {
   input: { aggregation-of
           { paper-basic-info:
             referee-report-dossier: }
          }
    output: list-of-selected-paper:
     event-on-termination: paper-selected;
     max-duration: 1 month;
      activation-time:time-point;
    activator:editor;
   steps: aggregation-of
           {reading-list;
            paper-selection-decision;
            write-list;}
   }
 }
```

The activities in SOS are described by the information objects (input or output) related to them, the activity's time duration, the agent who is responsible for the activity, and the steps of the activity. Based on the activity description, office tasks in SOS are supported by office evolutive rules. An office evolutive rule, specifying when to send a solicitation letter to a later referee is as follow:

```
<OER1> is-a office-evolutive-rule
{ at ( ending-time part-of referee-request) +1 month;
  if not exist referage such that
    ( referee part-of referage equal destination
      part-of letter-of-referage-request) and
     ( paper part-of referage equal paper part-of
       letter-of-referage-request);
  then send letter-of-solicitation-to-referee to
       destination part-of letter-of-referage-request }
```

2.4 Conclusions

To develop an office information system, the data representation, activity representation and task representation must be considered together. Among the developed models, form-based office information models have identified a structure which is able to couple data representation and activity representation together, but so far the form-based models are not able to represent nested data, therefore the activities upon the forms are not represented on the ground of office situation, it is therefore difficult to develop a cognitive problem solving process based on organizational semantics.

Based on this observation, one effort can be made to improve the data reference mechanism of a form-based model, then to introduce the techniques developed in the AI activity planning area to improve the performance of the form-based Office Information Systems.

Chapter 3.

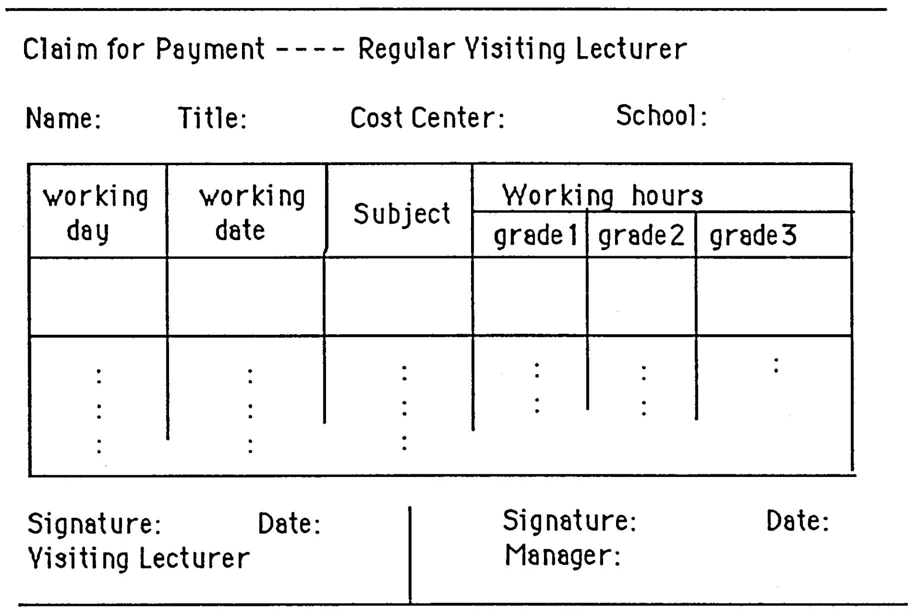
A Formbase System

This chapter presents a Formbase System. It is developed based on the earlier form-based office models such as FFM [Tsichritzis, D., 1982] and SOS [Bracchi, G. and Pernic, B., 1984]. Compared to the approaches of extending relational algebra to process the nested data such as [Mark A. Roth, et al, 1988], this system is developed based on the concepts of Office Information Systems, the data modellingmechanism is more semantic and easier to be accepted. Compared to the data modellingmechanism of Object Oriented Database Systems [Kim H. J., et al 1987], this system directly deals with ordinary forms and does not need inheritance relation and procedure attachment, therefore is easy to implement efficiently. The definition for a form type is based on aggregation and association abstractions which were used by SOS [Bracchi, G. and Pernic, B., 1984]. The predicates and the operations for the data inside the forms are based on the development of the form reference pattern language which is first introduced in [Liu. H., et al 1991]. A prototype of the system has been implemented on a Sun Workstation using Common Lisp.

3.1. Form Type Definition

Office forms are common information objects in an organization. Figure 3.1 is a claim for payment form, called RVL form, for a Regular Visiting

Lecturer in an institute. It is an ordinary type of office form. There are two levels of information in this form: the first level is constituted by the less changeable information such as 'Name', 'Title', and 'School'; the second level is constituted by the more dynamic information such as 'Subject', 'Working day' and 'Working hours'. Only two types of abstraction are needed to model such type of form, namely aggregation and association. ***Aggregation*** defines different subfields of a form, while ***association*** specifies groups of elements of the same type. By using aggregation and association abstractions, the RVL form can be represented as the following:



Claim for Payment ---- Regular Visiting Lecturer

Name: Title: Cost Center: School:

working day	working date	Subject	Working hours		
			grade1	grade2	grade3
: : :	: : :	: : :	: :	: :	:

Signature: Date:
Visiting Lecturer

Signature: Date:
Manager:

Figure 3.1

```
(define-form
  '(RVL
      (name)
      (title)
      (school)
      (cost-centre)
      (aggregation-of lecturer-signature
          (lecturer-signature)
```

```
        (date))
    (aggregation-of   manager-signature
        (manager-signature)
        (date))
    (association-of form-body
        (working-date)
        (working-day)
        (subject)
        (aggregation-of working-hours
            (grade1)
            (grade2)
            (grade3)))))
```

This representation is very straight forward and clear. Office forms can be generalized by recursively using aggregation and association abstractions. Its BNF definition is shown as the following:

```
<form-definition> : := (define-form <type-name> <form-type-tree> )

<form-type-tree>   : := (<attribute>) |
                    : := {aggregation-of <aggre-name> <form-type-tree> } |
                    : := {association-of <assci-name> <form-type-tree> } |
                    : := <form-type-tree> <form-type-tree>

<type-name>       : := symbol
<attribute>       : := symbol
<aggre-name>      : := symbol
<assci-name>      : := symbol
```

The inheritance relation is not needed in this definition though most of developed knowledge representation systems regard it as an essential abstraction means. This is not a surprise result since an inheritance relation does not have a wide semantic support.

3.2. Form Reference Patterns

Given the above definition of a form type, the form reference patterns can be composed by the form type and the names of the fields and the attributes, and their values. For example, the following pattern

```
(?rvl (instance-of  RVL)
      (name  Howard))
```

may be considered to refer to all the instances of the RVL forms with a constraint 'name = Howard'. And the pattern

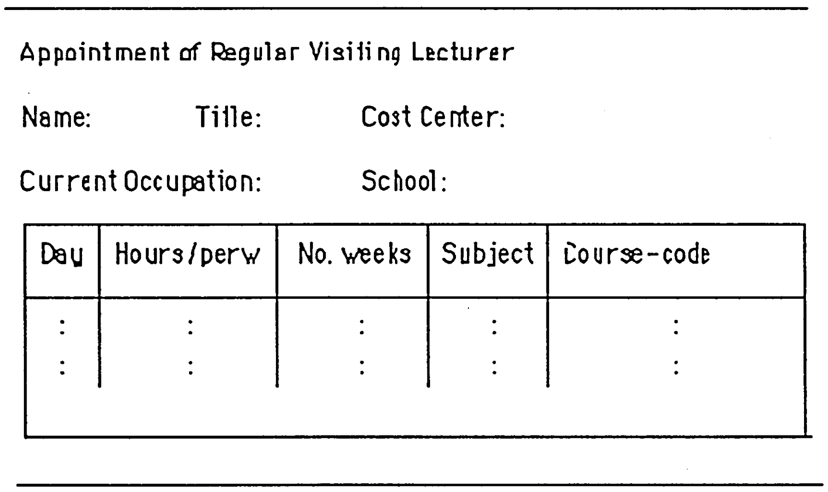
```
(?rvl  (instance-of RVL)
       (name  Howard)
       (lecturer-signature (date "3/12/90")))
```

may be considered to refer to all the instances of the RVL form with constraint 'name = Howard' and the date-of-signature = '3/12/90'. Notice here the 'lecturer-signature' field is an aggregation abstraction. The reference pattern may also use association abstractions. For example the following pattern is also clear. Besides the requirements for name and lecturer signature date, the pattern also has requirements for the subject that is taught on Monday.

```
(?rvl  (instance-of  RVL)
       (name Howard)
       (lecturer-signature
            (date "3/12/90"))
       (time-tab
            (subject software-tools)
            (working-day Monday)))
```

The 'time-tab' field is defined by an association abstraction. The constraints (subject "software tools") and (working-day "Monday") are upon the repeated group (or subform) 'time-tab'. Therefore, semantically it means that one of the subjects that the lecturer teaches is on Monday.

If the value inside a constraint is allowed to take variables, then the relations between different forms can be included into the form reference pattern. In this way a pattern language is defined. For example, if we have another form, called ARVL, in the institute for the Appointment of Regular Visiting Lecturer, as shown in Figure 3.2.


Appointment of Regular Visiting Lecturer

Name: Title: Cost Center:

Current Occupation: School:

Day	Hours/perw	No. weeks	Subject	Course-code
:	:	:	:	:
:	:	:	:	:

Figure 3.2

It can be defined as follows:

```
(define-form
   '(ARVL
        (name)
        (title)
        (school)
         (current-occupation)
        (cost-centre)
        (manager-signed)
         (association-of teaching-course
            (day)
(hours/perw)
 (number-of-weeks)
(subject)))
```

If we are interested in those RVL form instances such that for each of them, there is an ARVL form instance which has the same name, same cost centre, same subject, and is authorised by the same person, the form reference pattern can be written as the following:

```
(join ( (?rvl  (instance-of RVL )
              (name ?name)
             (cost-center ?cc)
              (time-tab (subject ?subject))
               (manager-sig (manager-sig ?man)))
           (?arvl  (instance-of ARVL)
                    (name ?name)
                   (cost-centre ?cc)
                    (manager-signed ?man)
                    (teaching-course (subject ?subject))))))
```

This pattern means for all the RVL form instances, there is an ARVL form instance which has the same name, same cost centre, same subject, and is authorised by the same person. The reference pattern for the office form is interpreted as an form type together with a set of constraints. A constraint can require an attribute of an office form have a special value, or require that there exists a special relationship between two attributes which could be inside the same form or inside different forms. For example the matching between the '?subject' of the RVL form, and the '?subject' of the ARVL form requires an equivalent relation between these two attributes. If a variable inside a constraint of a form pattern does not have a counterpart in other form patterns, the variable will make no sense.

3.3. Formalisation of Form Reference Pattern

Based on the above discussion, A Form Reference Pattern (FRP) can be defined by form types and constraints, shown as the following expression.

```
FRP = ( (?V1 T1 C11 C12 ... ... C1n)
        (?V2 T2 C21 C22 ... ... C2n)
           :
           :
         (?Vm Tm Cm1 Cm2 ... ... Cmn))
```

In this expression, ?V1, ?V2 and ?Vm are variables which represent form instances, T1, T2 and Tm specify the form types, and all the Cij (where i=1 ... m, and j=1 ... n) are constraints. A constraint consists of a value and a path from the top abstraction to the bottom abstraction where the value is located. The value can be an atom value, or a variable. More than one constraint condition can be specified if the field is an association abstraction. The variables inside a constraint are used to set up relational constraint between different forms. The matching pattern between RVL and ARVL forms is an example of a FRP:

```
( (?rvl  (instance-of RVL )
         (name ?name)
        (cost-center ?cc)
         (time-tab (subject ?subject))
         (manager-sig (manager-sig Susan)))
   (?arvl  (instance-of ARVL)
           (name ?name)
          (cost-centre ?cc)
           (manager-signed Susan)
           (teaching-course (subject ?subject)))))
```

The BNF definition for a form reference pattern is:

```
<FRP>             : := ( <Form-var> <Form-type-spec> <FRP-body> )
```

```
<FRP-body>          : := Empty |
                    : := <single-FRP> | <single-FRP> <FRP-body>
<single-FRP>        : := <constraint-cell>
                    : := ( <field-name> <single-FRP>)
<constraint-cell>   : := (<field-name> <value>)
<Form-type-spec> : := (instance-of  <form-type>)
<form-type>         : := symbol
<field-name>        : :=symbol
<Form-var>          : := <var>
<value>             : := symbol | <var>
<var>               : := ?symbol
```

3. 4. Five-Key Allocation for Values inside a Formbase

A Form Reference Pattern defines a set of form instances inside a formbase in which we are interested, but to manipulate the values inside a formbase, the system not only needs to know the set of form instances of our interest, but also needs to be able to allocate exactly where the values are. Based on the form reference pattern language, this section develops a five-key system which is able to allocate a value inside a formbase.

If we want to manipulate a set of values, they are first inside a set of form instances of interest. Once the set of form instances of interest are allocated, the system needs to know which form type, which abstraction and which attribute of the abstraction that the values are located. If the abstraction is an association abstraction, more constraints upon the repeated group also need to be specified. Therefore the system needs five keys to allocate values inside the formbase. These keys are represented respectively as **:form, :type,**

:subarea-path, :sub-cond and **:attribute.** The key **:forms** uses the FRP to define the form instances of interest. The **:type** key specifies which type of the form instances that the operator/predicate is applied to.

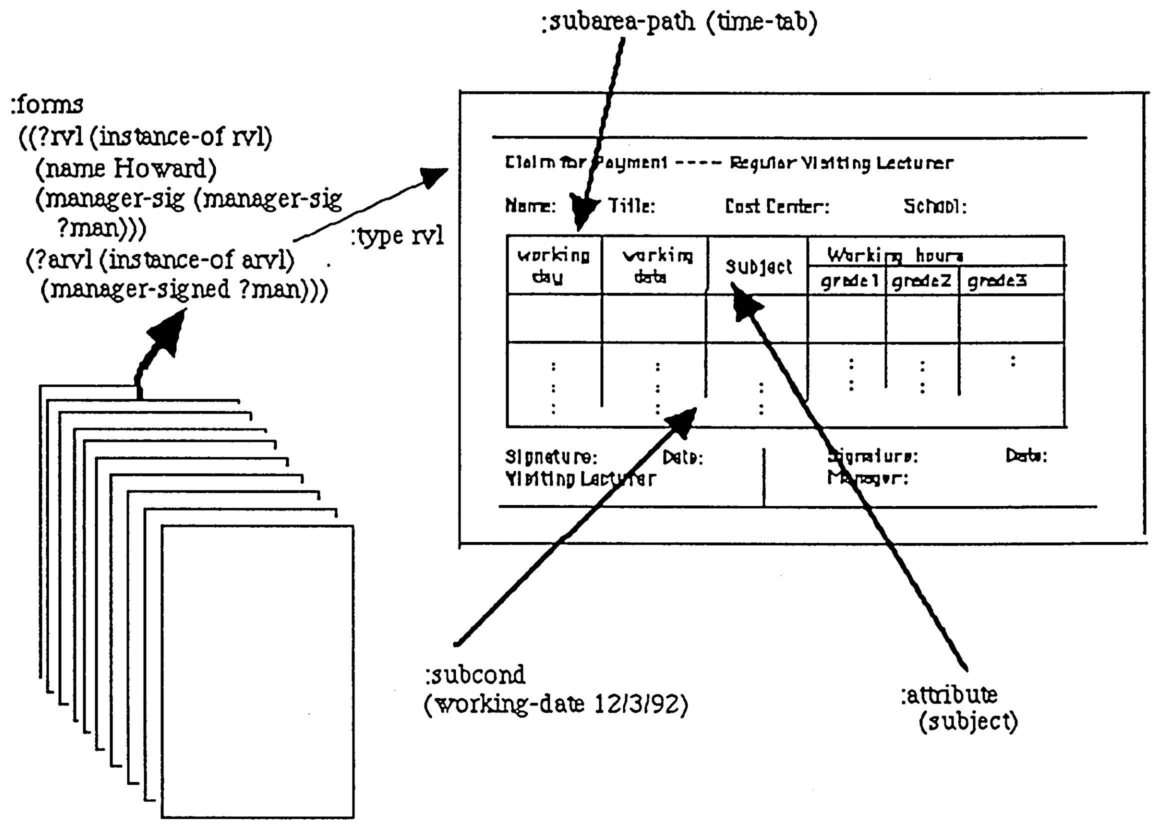


Figure 3.3

An attribute is inside an abstraction, for example the 'subject' attribute is inside 'teaching-course' abstraction in ARVL definition, or it can directly belong to the root instance, for example the 'name', 'title' attributes directly belong to the root instance in ARVL forms. An abstraction is called the **parent abstraction** of an attribute if this attribute directly belongs to it. In order to manipulate the values, **:subarea-path** is used to allocate the parent abstraction of the attribute. If the attribute is a son of the root form instance, then the **:subarea-path** key needs not to be specified. If the parent abstraction of the attribute is an association abstraction, then the system faces a table of data, the **:sub-cond** key is used to specify the set of rows to which the operator/predicate is applied. The **:attribute** key specifies the attribute that the operator/predicate is applied.

Figure 3.3 gives an example of allocating values inside a form instance. Suppose we are going to manipulate the 'subject' attribute of the verified 'rvl' forms (Fig. 3.1) whose 'name' attribute value is 'Howard'. The **:forms** key is:

```
((?rvl (instance-of rvl)
      (name Howard)
        (manager-signature (manager-signature  ?man)))
  (?arvl (instance-of arvl)
        (name Howard)
         (manager-signed ?man)))
```

The 'arvl' form (Fig 3.2) is needed to verify the 'rvl' form. The value of **:type** key is 'rvl' to indicate which type of form that the operator is going to be applied to. Since the 'subject' attribute is inside 'time-tab' association abstraction, and it is the parent abstraction of it, the **:subarea-path** key is (time-tab). Finally because 'time-tab' abstraction is an associate abstraction, the **:sub-cond** is specified. The whole allocation process is shown in Figure 3.3.

3. 5. Query upon Forms

Based on the above allocation system, queries for form base can be defined. Query function can use the five keys for determining the scope of the form instances. Therefore, the query function is defined as follows:

```
(formbase-query :forms :type :subarea-path :sub-cond :attribute)
```

Except the :form key, all the other keys are optional as long as the query is meaningful. For example,

```
(formbase-query :forms ((?rvl (instance-of rvl)
                              (name Howard)
                               (manager-sig (manager-sig  ?man)))
                        (?arvl (instance-of arvl)
                               (name Howard)
                                (manager-signed  ?man))) )
```

searches all the 'rvl' and 'arvl' forms whose name attribute values are 'Howard' and are signed by the same manager. If we are only interested in the instances of 'rvl' forms under the same condition, then the query is:

```
(formbase-query :forms ((?rvl (instance-of rvl)
                              (name Howard)
                               (manager-sig (manager-sig  ?man)))
                        (?arvl (instance-of arvl)
                               (name Howard)
                                (manager-signed ?man)))
                :type rvl)
```

The :type key is added. If we are interested in the value of the 'school' attribute of the 'rvl' form under the same condition, then the query is:

```
(formbase-query :forms ((?rvl (instance-of rvl)
                              (name Howard)
                               (manager-sig (manger-sig ?man)))
                        (?arvl (instance-of arvl)
                               (name Howard)
                                (manager-signed  ?man)))

                :type rvl
                :attribute school )
```

If we are interested in the 'subject' attribute of the 'rvl' form, since it is located in the 'time-tab' abstraction, the :subarea-path key is needed.

```
(formbase-query :forms ((?rvl (instance-of rvl)
                                (name Howard)
                                 (manager-sig (manager-sig ?man)))
                         (?arvl (instance-of arvl)
                                  (name Howard)
                                   (manager-signed  ?man)))
                :type rvl
                :subarea-path (time-tab)
                :attribute subject )
```

This query returns all the subject values of the 'rvl' form instances. If we are only interested those 'subject' attribute values which are taught on Monday, then the query takes the following form:

```
(formbase-query :forms ((?rvl (instance-of rvl)
                                (name Howard)
                                 (manager-sig (manager-sig ?man)))
                         (?arvl (instance-of arvl)
                                  (name Howard)
                                   (manager-signed  ?man)))
                :type rvl
                :subarea-path (time-tab)
                :sub-cond (working-day Monday)
                :attribute subject )
```

3. 6. Predicates upon Forms

Predicates upon forms can be defined. Currently the implementation of the system supports **exists, some-equal, every-equal, subgroup,** and **exist-equal** predicates. Similarly every predicate can take the five keys as long as they are meaningful. For example, **exists** predicate can use **(exists :forms)** to check if there are forms inside the form base which match the form reference pattern. Or it can use **(exists :forms :type :attribute)** to check if the value of the attribute of the type of the form instances exist. Similarly **(exists :forms :type :subarea-path)** checks if the values of the sub-area/sub-abstraction exist, or checks a subset of the values of the sub-area by using **(exists :forms :type :subarea-path :sub-cond)**, or checks the values of a certain attribute of a certain sub-area of certain type by using **(exists :forms :type :subarea-path :sub-cond :attribute)**.

For the predicates which take two set of values, such as the predicates like equal and subgroup, two sets of keys are used. They are labelled as **:formsl, forms2, :type1, type2, :subarea-path1, subarea-path2 :sub-cond1, sub-cond2, :attribute1** and **:attribute2**. In the following some examples of these predicate are given out.

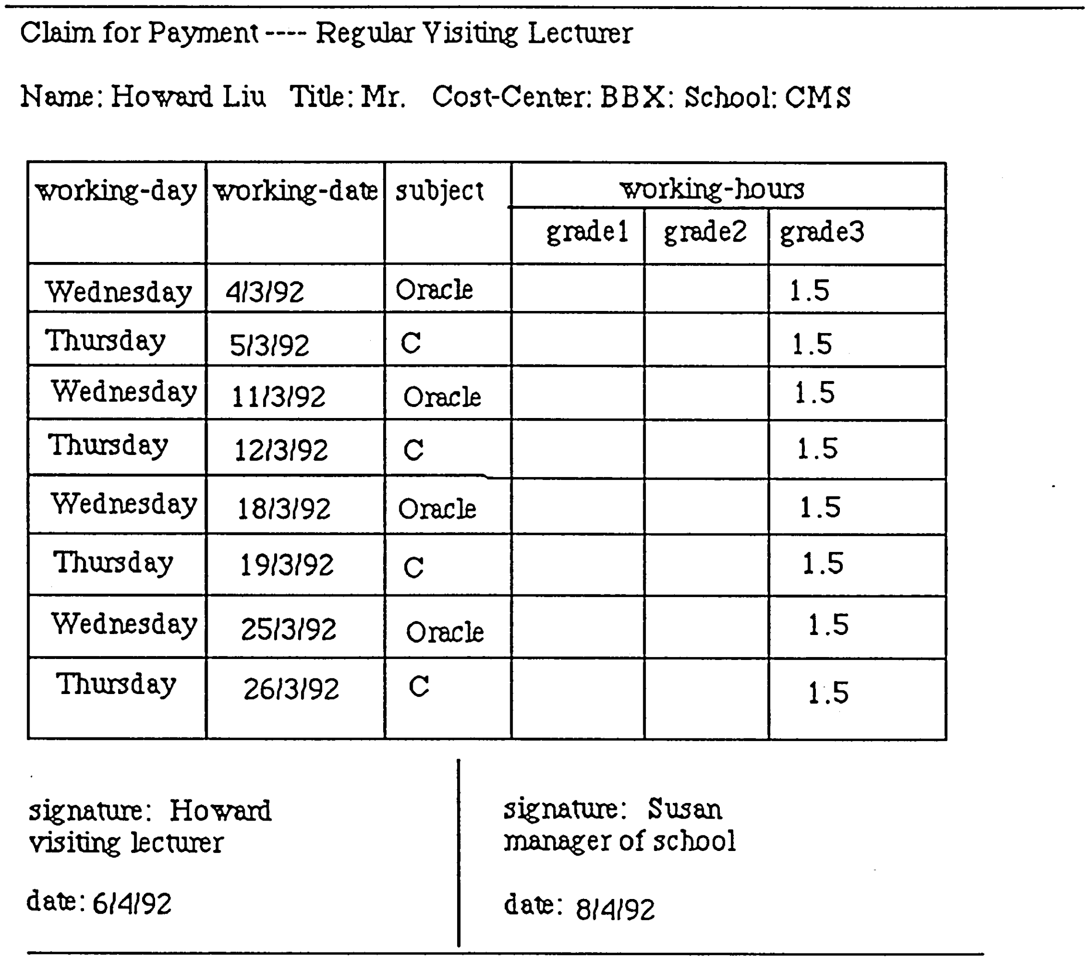
If the form of Fig. 3.4 and Fig. 3.5 exist inside the form base, then the following expression should return true.

```
(exists  :forms ( (?rvl  (instance-of RVL )
                         (name ?name)
                        (subject ?subject)
                        (cost-center ?cc)
                         (manager-sig (manager-sig Susan)))
                  (?arvl  (instance-of ARVL)
                            (name ?name)
                            (cost-centre ?cc)
                            (manager-signed Susan)
                            (teaching-course (subject ?subject)))))
```

Predicate some-equal returns true if some of the values of the query are equal to the value, for example,

```
(some-equal
         :forms ((?rvl (instance-of rvl)
                        (name Howard)
                        (school CMS)))
          :subarea-path (time-tab)
          :attribute subject
         :value Oracle)
```

returns true since in Figure 3.4, Howard does teach Oracle. The predicate 'every-equal' returns true only if every values of the of the query equals the value. Therefore, based on the facts in the form of Fig. 3.4, the following expression returns false, since not every subject attribute takes the value of 'Oracle'.



Claim for Payment ---- Regular Visiting Lecturer

Name: Howard Liu Title: Mr. Cost-Center: BBX: School: CMS

working-day	working-date	subject	working-hours		
			grade1	grade2	grade3
Wednesday	4/3/92	Oracle			1.5
Thursday	5/3/92	C			1.5
Wednesday	11/3/92	Oracle			1.5
Thursday	12/3/92	C			1.5
Wednesday	18/3/92	Oracle			1.5
Thursday	19/3/92	C			1.5
Wednesday	25/3/92	Oracle			1.5
Thursday	26/3/92	C			1.5

signature: Howard
visiting lecturer

date: 6/4/92

signature: Susan
manager of school

date: 8/4/92

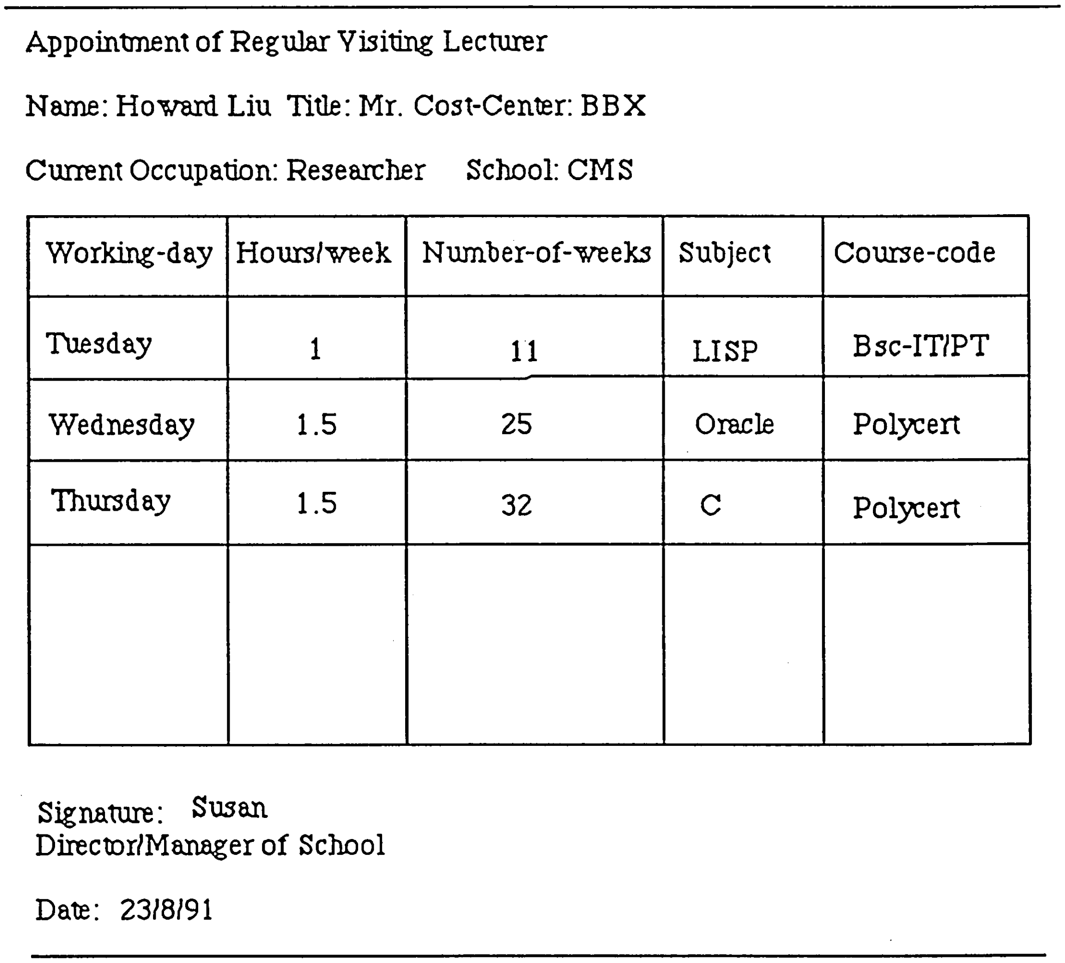
Figure 3.4

```
(every-equal
          :forms ((?rvl (instance-of rvl)
                        (name Howard)
```

```
                    (school CMS)))
        :subarea-path (time-tab)
        :attribute subject
        :value Oracle)
```

However, the following expression should returns true, since the working hours of every subject is 1.5hrs.

```
(every-equal
        :forms ((?rvl (instance-of rvl)
                     (name Howard)
                     (school CMS)))
        :subarea-path (time-tab (working-hours))
        :attribute grade3
        :value 1.5)
```



Appointment of Regular Visiting Lecturer

Name: Howard Liu Title: Mr. Cost-Center: BBX

Current Occupation: Researcher School: CMS

Working-day	Hours/week	Number-of-weeks	Subject	Course-code
Tuesday	1	11	LISP	Bsc-IT/PT
Wednesday	1.5	25	Oracle	Polycert
Thursday	1.5	32	C	Polycert

Signature: Susan
Director/Manager of School

Date: 23/8/91

Figure 3.5

The predicate **subgroup** returns true if all the form instances which satisfy the first set of keys also satisfy the second set of keys. For example,

```
(subgroup :forms1 ((?rvl (instance-of rvl)
                         (name Howard)))
          :subarea-path1 (time-tab)
          :sub-cond1 (subject Oracle)
          :forms2 ((?arvl (instance-of arvl)
                          (name Howard)))
          :subarea-path2 (teaching-course)
          :sub-cond2 (working-day Wednesday) )
```

returns true based on the fact inside the form which is shown in Fig. 3.4. This true value says that all the Oracle courses are taught on Wednesday.

The predicate **exist-equal** returns true if there are values of the first set of keys equal to the values of the second set of keys. Therefore, the following expression should return true based on the facts inside the forms of Fig. 3.4 and Fig. 3.5.

```
(exist-equal
          :forms1 ((?rvl (instance-of rvl)
                         (school CMS)))
          :subarea-path1 (time-tab)
          :attribute1 subject
          :forms2 ((?arvl (instance-of arvl)
                          (school CMS)))
          :subarea-path2 (teaching-course)
          :attribute2 subject)
```

3. 7. Operations Upon Form Base

The operations that a form base needs are similar to the operations that are needed by a data base system. They are 'set', 'delete' and 'modify' values of the attributes of form instances. The same set of keys are used to determine the attributes to which the operator is applied. Plus operation type and the value for setting and modifying, an operation takes the following format:

```
(Form-Op : forms
         : type
         : subarea-path
         : subcond
         : op-type
         : attribute
         : value )
```

The key **: op-type** specifies the type of the operator, it could be **set-value, delete-value, modify-value** and **delete-subform**. The meaning of the operations will be explained in the later examples. The key **: value** specifies the value that is set to or is modified to.

For example,

```
    (Form-Op : forms ((?rvl (instance-of rvl)
                            (name Howard)))
             : Op-type modify-value
             : attribute title
             : value Dr.)
```

will modify the value of 'title' attribute of the form instance shown as Fig. 3.4 to 'Dr.'. In this operation, the key **: subarea-path** and **: subcond** need not be specified since 'title' attribute is a direct attribute of the form instance. If on the 19th of March, the lecturer actually taught 1.5 hours' Pascal language rather than C, then we need the following modification operation.

```
    (Form-Op  : forms ((?rvl (instance-of rvl)
                             (name Howard)
                             (lecturer-sig (date 6/4/92))))
              : subarea-path (time-tab)
              : subcond ((working-date 19/3/92))
              : Op-type modify-value
```

```
                : attribute subject
                : value Pascal)
```

This operation first finds the form instance whose name attribute value is 'Howard' and is signed by the lecturer on 6th of April, 1992. It then finds the subarea of the attribute which is 'time-tab' (see RVL definition) in this form. This subarea is a subform of the form instance, it therefore has sub-condition ((working-date 19/3/92)) to specify which row the operator is applied to. Notice that sub-conditions only can be specified when the subarea is a subform which corresponding to an association abstraction. If, for example, we know on the 19th of March, the lecturer actually taught 3 hours rather than 1.5 hours, then we need the following modification operation.

```
    (Form-Op  : forms ((?rvl (instance-of rvl)
                             (name Howard)
                             (lecturer-sig (date 6/4/92))
                             (time-tab (working-date 19/3/92))))
               : subarea-path (time-tab (working-hours))
               : Op-type modify-value
               : attribute grade3
               : value 3)
```

For this operation, the condition ((working-date 19/3/92)) is specified in the **forms** key since the subarea of this operation is 'working-hour' which is a subfield of 'time-tab'.

If the operation type is not **set-value** or **modify-value**, then we do not need to specify **: value** key. If the operation is **delete-subform**, then we even need not to specify **: attribute** key. For example,

```
    (Form-Op  : forms ((?rvl (instance-of rvl)
                    (name Howard)
                    (lecturer-sig (date 6/4/92))))
        : subarea-path (time-tab)
        : subcond ((subject Oracle))
        : Op-type delete-subform)
```

will delete all the rows whose 'subject' attribute has value 'Oracle' in the RVL form instance whose name is 'Howard' and is signed by the lecturer on 6th of April.

In addition to the above operations, the form base management system also supports **add-subform** operation which add a row into a subform of a form instance. A query to form base is to find all the form instances which satisfy the constraints that are specified by a form reference pattern. In the current implementation, function **(list-form-instances :forms)** searches the form base and lists all the subfields, subforms and their attribute values.

3. 8. Features of Form Reference Patterns

This section presents a detailed discussion about the features of the form reference patterns. A form reference pattern is a set of constraints upon a form type. For example, if we have many goods containers which have different colours, weight, and are made in different places. Then

```
(?container (instance-of container)
            (colour red))
```

means that the ?container variable represents all the containers whose colour are red. And

```
(?container (instance-of container)
                (colour red)
                (made-in England))
```

means that the variable ?container represents all the containers whose colour are red and are made in England. To further study the features of the patterns, in the following the patterns are compared with first order predicates. If the above examples are represented by predicates, the following relation is defined first:

(container container-id colour made-in-place weight)

Then predicate (container ? red ? ?) represents all the containers whose colour is red, and (container ? red England ?) represents all the containers whose colour are red and are made in England. From this example, we can see that in a form reference pattern, the order of the constraints is not meaningful, but in a predicate the order of the constraint values is important since their semantic meaning is actually given by their position in the relation.

3.8.1 Why not Predicates

A predicate, a functor or a relation represent the relationship between data through a row of positions. These positions are remembered by the processing program. This representation method is concise and expressive, and has been used by almost all the AI systems to describe information and knowledge. However to represent knowledge in term of their positions inside a row implies that the semantic dependent relation of data is fixed and the working context of the activities to access these data are predefined.

So the methods are effective only if the system which we are going to describe is closed and small. In an organization, a meaningful information object is usually processesed by different departments which have different working contexts (or viewpoints). It is then difficult to use the relational patterns to effectively represent the information objects.

For example the RVL form (Fig. 3.1) is an ordinary office form. Like all the other forms, it provides and records information for the related information processing activities. A visiting lecturer fills his/her work to claim the payment, the manager certifies the hours that the lecturer has claimed, and the cashier department processes the payment. However, it also can be noticed that the manager can not certify a form without a name, and the cashier department can not process a form without the signature of the manager. So, the data integration of a form implies not just storing and providing data, but also implies a sequence of activities upon the form. This implication certainly can not be afforded by the relational representation patterns.

The activities upon a form are usually located in different departments of an organization and have different working contexts. They have different interest for the data inside a form. For example, the cashier department will concern information such as cost centre, while the visiting lecturer has no interest for it at all. An important advantage of the form reference pattern is that it could form different concerns (viewpoints) based on one form type. For example,

```
(?rvla (instance-of RVL)
      (Name Howard)
      (School CMS)
```

```
                    (time-tab (Day Monday)))                    (a)
```

refers to all the RVL instances whose Name is Howard, School is CMS, and the Day is Monday. It concerns something which is connected with a specific lecturer in the school, and Monday. The following reference pattern reflects however another concern.

```
(?rvlb (instance-of RVL)
     (School CMS)
     (Manager-signature Susan))                    (b)
```

This reference pattern concerns the connection of the Manager of the school and the RVLs. The interesting thing is though they reflect different concerns, they are defined based on the same information object! This is the way which the information is organized in practice. A form or a document does not provide information for just one working context, but potentially for all the working contexts related to it.

3.8.2 Problem of Unification

Because the form pattern language is able to represent the association abstraction, the unification process for first order predicate calculus is not applicable to it. For example, the following two first-order predicates can be unified.

```
(container   ?   red      ?       ?)
(container   ?   red   England    ?)
```

They can correspond to the following form reference patterns.

```
(?container (instance-of container)
            (colour red))

(?container (instance-of container)
            (colour red)
            (made-in England))
```

Because the form patterns in this example do not have association constraints, they are consistent with the unification process (the latter expression represents a subset of the former). If a form reference pattern contains association abstraction, then it can not be consistent with the unification process. For example,

```
      (?rvl1 (instance-of RVL)
             (School CMS)
             (Subject Software-tools))                  (f)
and
      (?rvl2 (instance-of RVL)
             (School CMS)
             (Subject Information-system))              (g)
```

are two form reference patterns. Pattern (f) refers to all the visiting lecturers who are teaching software tools, pattern (g) refers to all the lecturers who are teaching information systems. Their corresponding relational patterns are:

```
      (RVL  ?  CMS     Software-tools     ? .. ..  ?)     (h)
and
      (RVL  ?  CMS  Information-system   ? .. ..  ?)     (j)
```

According to unification procedure, pattern (h) and (j) can not be unified because the subject attribute has different values: 'software-tools' and 'information system'. However, the form patterns (f) and (g) may represent a same set of instances of the RVL form since a lecturer may teach both of the subjects 'information system' and 'software tools'. In fact, the relation between the two sets can not be judged from the syntax level. They could be one set, or absolutely exclusive. The judgement only can be made with the help of the semantics.

Chapter 4.

An Implementation of the Formbase System

This chapter presents an implementation of the formbase system. The implementation of the form base system concerns three main problems. They are:

- the internal representation of form type and form instances;
- the processing of a single form reference pattern, that is the query upon one form type;
- and the processing of a joined form reference pattern which is joined query upon more one forms.

The current prototype of the formbase is implemented in a UNIX environment at Sun-3 Workstation using Common Lisp. The data of forms can be persistently stored in the Unix filing system.

4. 1. The Internal Representation Forms

The semantic data model which the Formbase System has used to represent office information objects implies that it is necessary be able to model the association relation between a form (an entity) and its associated subforms which have an arbitrary number of instances. Such an association relation is very common for an office form, since there are always some messages in a form more changeable than the others though they are bound together. The SOS office model [Bracchi and Pernic, 1984] has the ability to define the type of a form, but it has not proposed an internal representation and a proper retrieval language.

The representation and retrieval strategy that the Formbase System uses is developed based on the assumption that once an office form type has been defined, its subforms and attributes (association relation and aggregation relation) are all fixed. Therefore the type of an office form is easily stored. Therefore, every input of an instance of a form type is to model a copy of the form type. The retrieval of the pattern language can use the required values of the attributes as a set of constraints upon all the instance occurrences of the form type. Therefore the retrieval process becomes a constraints satisfaction process.

4.1.1 Internal Representation of Form Type

A form contains attributes, aggregation abstractions or association abstractions. Each aggregation abstraction and association abstraction again can have attributes, aggregation abstractions and association abstractions. Therefore, a form type can be represented as a tree, called **type-tree**. Figure 4.1 is the type-tree for RVL form type. Every leaf of the tree is a **schema** which only contains a set of attributes, that is, it is the parent of a set of attributes. In Figure 4.1, the root schema RVL contains the attributes that belong to the top level abstraction of the form type. The other attributes of the form belong to their own parent, and are related to the root through **association-of** and **aggregation-of** relations.

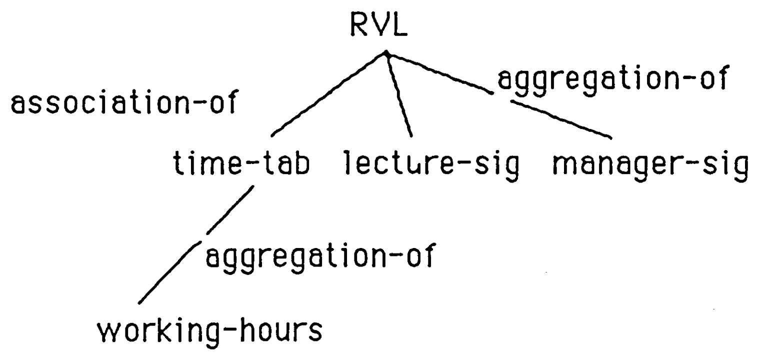


Figure 4.1

So, Figure 4.1 and Figure 4.2 illustrate the representation of the definition of the RVL form (Fig. 3.1). Since in a tree every son can only has one parent, the relation between the nodes can easily represented by using the following relation pattern:

(schema-name assic-parent aggre-parent)

For example, the form type of Figure 4.1 can be represented by the following instances of the pattern:

```
(manager-sig   nil   rvl)
(lecturer-sig    nil  rvl)
(time-tab   rvl  nil)
(working-hours  nil   time-tab)
```

It is even simpler to represent schemas. The schemas in the type tree of Figure 4.1 are shown as Figure 4.2. Obviously simple relations can represent the schemas.

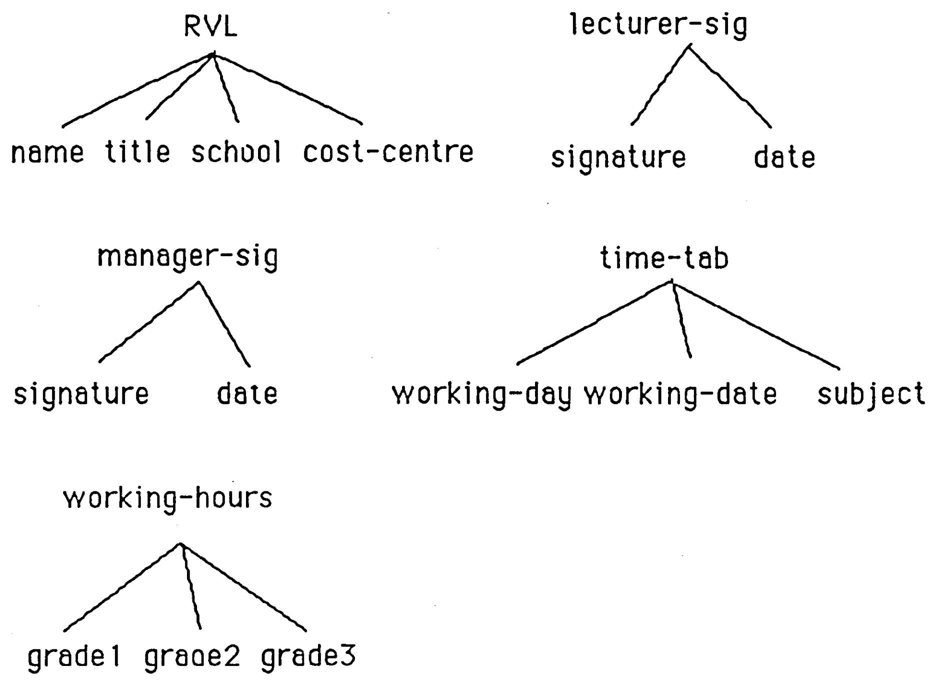


Figure 4.2

4.1.2 Internal Representation of Form Instance

Every form type may have an arbitrary number of instances. How should one form instance be represented?

In current prototype system, just as one type-tree represents a form type, one **instance-tree** represents a form instance. The schemas which are used to represent an instance-tree are the occurrences of the schemas in the type representation of the form. They are automatically generated and managed by the system, the names of these schemas are meaningless. For example Figure 4.4 is a representation of an instance of the RVL form. The names of the schemas are generated by the system. In this example, G309 is an instance of RVL schema in the Figure 4.1, if its name value is 'Howard', title is 'Mr.' school is 'CMS' and cost-centre is 'bbx', then it is a relation shown as

Figure 4.3. Correspondingly G310, G311, G312, and G313 are instances lecturer-sig, manager-sig, time-tab and working-hours. Obviously, G309, G310 and the others, are instances of the schemas in the type-tree. The connections of them constitute an intance-tree of an instance of the RVL form.

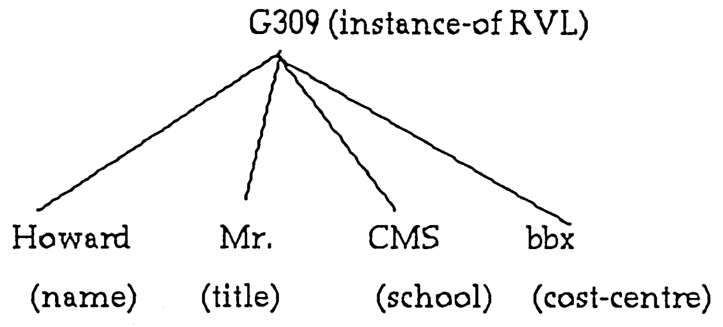


Figure 4.3

In the tree representation, the *subfield-of* relation is related to the aggregation abstraction in the form type definition, while the ***subform-of*** relation is related to association abstraction in the form type definition. In the Formbase System, an instance tree is represented by the following relational pattern:

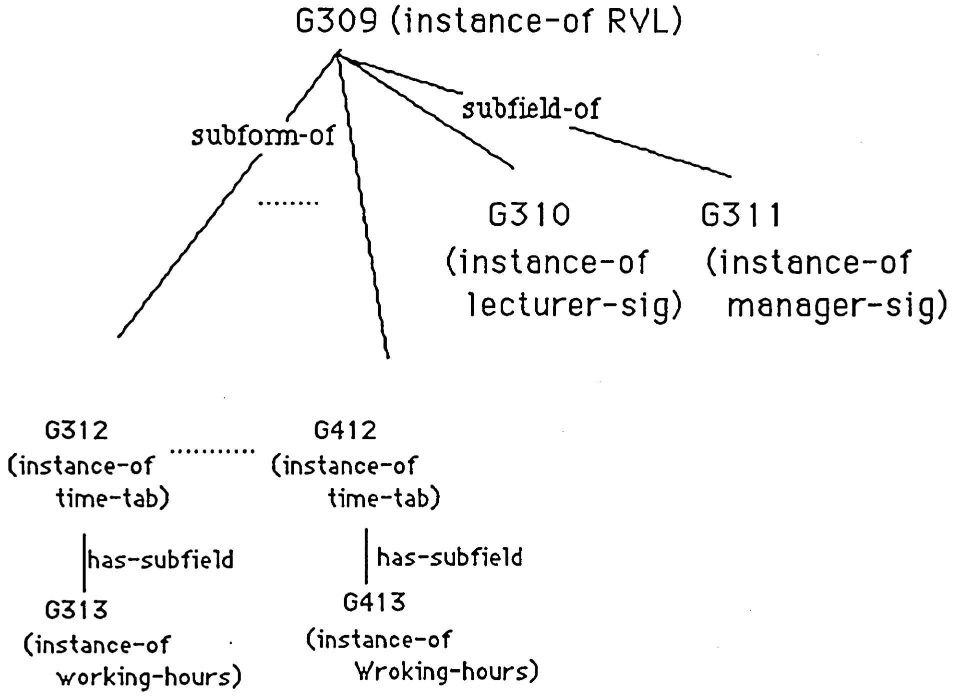
(instance subform-of subfield-of)

Therefore, Figure 4.4 can be represented by the following instances of the relational pattern:

```
(G310   nil    G309)
(G311   nil    G309)
(G312   G309    nil)
(G313   nil    G312)

      :
      :
```

```
(G412   G309     nil)
(G413     nil   G412)
```

How many subforms that a form instance such as G309 may have (such as G312, G412) is arbitrary.



G309 (instance-of RVL)
subform-of
subfield-of
G310 (instance-of lecturer-sig)
G311 (instance-of manager-sig)
G312 (instance-of time-tab)
G412 (instance-of time-tab)
has-subfield
has-subfield
G313 (instance-of working-hours)
G413 (instance-of Wroking-hours)

Figure 4.4

4. 2. Form Reference Pattern Processing

A Form Reference Pattern uses more than one form type to refer to a portion of data of interest in the formbase. In system implementation, we call the pattern which uses more than one form type **joined form reference pattern,** and the pattern which only uses one form type **single form reference pattern**. The semantic difference between joined form reference pattern and single reference pattern is illustrated in Figure 4.5 and Figure 4.6. A single form reference pattern does not contain variables and is upon one form type. While a joined form reference pattern contain variables and is upon more than one form type.

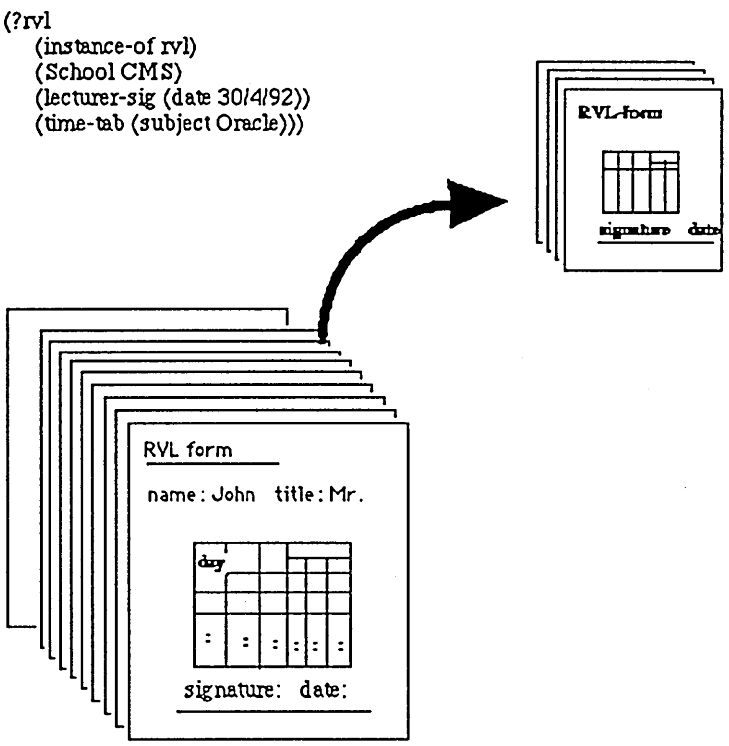
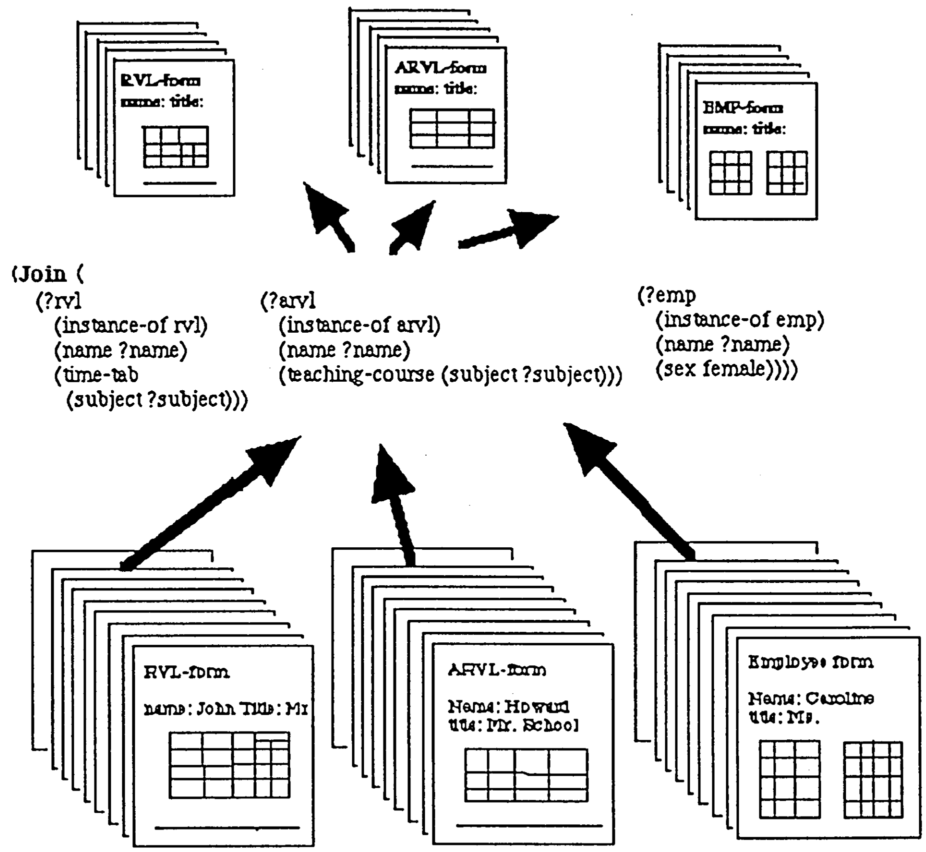


Figure 4.5

In order to determine whether a joined form reference pattern can be satisfied by the formbase, it is necessary to decompose it into many single form reference patterns, then to consider the relations between the single reference patterns. The algorithm for processing a joined form reference pattern has three steps: the first step is to abstract all the variables and decompose the joined pattern into many single form reference patterns. The second step is to process the single form reference patterns and find the instances of every type which satisfy the constraints of the pattern after the variables have been filtered. In the third step, the relational constraints which are identified in the first step are applied to the instances which are found in the second step. The results are the final set of instances that meet the requirements.



```
(Join (
   (?rvl                        (?arvl                                        (?emp
     (instance-of rvl)            (instance-of arvl)                            (instance-of emp)
     (name ?name)                 (name ?name)                                  (name ?name)
     (time-tab                    (teaching-course (subject ?subject)))         (sex female))))
       (subject ?subject)))
```

Figure 4.6

There are four data structures used in this algorithm. One is called 'var-list', one is called 'instan-set', one is called 'join-list', and the other is called 'production-list'. The 'var-list' data structure is generated in the first step, the 'instan-set' is generated in the second step, then both of them compose the 'join-list' which is used in the third step. The 'production-list' record the join results.

4.2.1 First Step ---- Filter Variables

The first step of the algorithm abstracts all the variables from the joined form reference pattern, and generates single form reference patterns and the 'var-list' data structure.The single form reference pattern will be processed in the second step. The 'var-list' will be used to form the 'join-list' in the third step. An element of 'var-list' is a list which contains a variable name and the **Attribute Path** of this variable. An **Attribute Path** is composed by the subarea-path of the parent abstraction and the attribute name. For example, (?subject (teaching-course (subject))) is an element of 'var-list' which contains a variable '?subject' and an attribute path '(teaching-course (subject))'. A 'var-list' is defined for one form type, it contains all the variables and their attribute paths inside the joined query. For example, if we have a form reference pattern as the following:

```
    (?rvl
           (instance-of rvl)
           (name Howard)
           (time-tab (subject ?sub))
           (manager-sig (date ?date)))
```

Then, after the first step, the program generates a single form reference pattern:

```
        (?rvl
          (instance-of rvl)
          (name Howard))
```

and a 'var-list':

```
( (?sub (time-tab (subject)))
  (?date (manager-sig (date))) )
```

4.2.2. Single Form Reference Pattern Processing

A **single-form-query** function is designed based on the following assumptions: every single form reference pattern is composed by a form type specifier such as (instance-of rvl), and a list of constraints such as (school CMS), (lecturer-sig (date 30/4/92)) and (time-tab (subject Oracle)). Every constraint is composed by a value and an **Attribute Path** (AP), for example the constraint (lecturer-sig (date 30/4/92)) is composed by the attribute path <u>(lecturer-sig (date</u> and the value '30/4/92'. The value of a constraint specifies the specific value of interest, the attribute path specifies which subform or subfield the attribute is allocated. The following is a single form reference pattern:

```
 (?rvl  (instance-of rvl)
      (school CMS)
      (lecturer-sig  (date  30/4/92))
      (time-tab  (subject  Oracle)
                (working-day Monday)
                (working-hour (grade3 1))))
```

To process a single form pattern is a constraint satisfaction process. The system first process the head of every attribute path which is underlined in this example, they are 'School', 'lecturer-sig' and 'time-tab'. If the head is an attribute, for example 'School', it will be put into a set called **attri-list,** if the head is an aggregation abstraction, for example 'lecturer-sig', it will put into a set called **aggre-list,** if the head is an association abstraction, for example 'time-tab', it will be put into a set called **assci-list**. Therefore after the initial process, we have:

```
attri-list: ((school CMS))
aggre-list: ((lecturer-sig  (date  30/4/92)))
assci-list: ((time-tab  (subject  Oracle)
                    (working-day Monday)
                    (working-hour (grade3 1))))
```

Obviously, the constraints for attributes consist of the first set of constraints for the top level instances in the instance tree.Therefore they are first applied to search out an initial set of instances, called **Init-Set**. In the **Init-Set**, the constraints of the attri-list are satisfied.After having **Init-Set, aggre-list** and **assci-list**, The Formbase System uses another function, called **instan-constraints-query**, to carry on the constraints satisfaction process against every instances in **Init-Set**. The **instan-constraints-query** function is defined as follows:

```
(instan-constraints-query instance attri-cons   aggre-cons assci-cons)
```

The 'instance' in the function specifies the instance submitted to the instance constraints satisfaction process, 'attri-cons' specifies the constraints upon attributes, 'aggre-cons' specifies the constraints upon aggregation abstractions, and the 'assci-cons' specifies the constraints upon association abstractions. The constraint satisfaction process checks the instance inside **Init-Set** one by one, if an instance satisfies the constraint conditions, then it will be kept, otherwise it will be deleted from the **Init-Set**. Therefore, in above example if there is an instance G309 which satisfies the (School CMS) constraint, then the system will have a function call:

```
(instan-constraints-query  G309  nil
 ((lecturer-sig  (date  30/4/92)))
 ((time-tab  (subject  Oracle)
            (working-day Monday)
            (working-hour (grade3 1)))))
```

To check if an instance satisfies the constraints, the process distributes the constraints to different levels of its instance tree. The distribution process searches out the subfields or the subforms of the instance for which the constraint are specified. Then there are two other functions to process the constraints upon aggregation and association respectively. They are:

(aggre-constraints instan type cons)

and

(assci-constraints instan type cons).

The 'instan' is the subfield instance or the subform instance of the corresponding abstraction types which the constraints are specified. Suppose the instance tree of the G309 is as shown in Figure 8. Then for the above example, the system will have:

```
(aggre-constraints G310 lecture-sig
                    ((date 30/4/92)))

(assci-constraints G312 time-tab
       ((subject  Oracle)
        (working-day Monday)
        (working-hour (grade3 1)))

(assci-constraints G412 time-tab
       ((subject  Oracle)
        (working-day Monday)
        (working-hour (grade3 1)))
```

These functions again will first process every head of the constraints (underlined) to construct the attri-list, aggre-list and assci-list at this level, then call the **instan-constraints-query** function recursively. For example, to process G412 in above, another call to **instan-constraints-query** will be made as the following:

```
(instan-constraints-query G412
  ((subject Oracle) (working-day Monday))
  ((working-hour (grade3 1)))
  nil)
```

This function first check if G412 satisfies the constraints upon its attributes 'subject' and 'working-day', if it does not, then G412 fails to satisfy the constraints. If it does, the constraint upon its subfield G413 will be further checked by a recursive function call:

```
(aggre-constraints G413 working-hour
            ((grade3 1)))
```

It will again call instan-constraints-query function. In this way, the constraints satisfaction process goes on and on.

Every single form reference pattern generates an 'instan-set'. An 'instan-set' contains all the form instances of one form type in the joined form reference pattern.

4.2.3 Third Step ---- Join

The third step of the algorithm is to join the query results of the second step by checking the relational constraints required by the variables. This step is similar to the join-pattern processing of Forge's Rete algorithm [Forge, C., 1982]. The join pattern net for the join of the form instances is shown in Figure 4.8, At the top of this join net, the input of every node is a 'join-list'. A 'join-list' is a list which contains form instance and the 'var-list' of a form type. For example, if 'instan402' and 'instan411' are the instances of form type 'rvl' which satisfy the query:

```
(?rvl
   (instance-of rvl)
  (manager-sig
     (manager-sig Susan)))
```

The the 'join-list' for pattern

```
        (?rvl  (instance-of RVL )
                 (name ?name)
                (subject ?subject)
                (cost-center ?cc)
                (manager-sig
                   (manager-sig Susan)))
```

is:

```
( (instan402
    ((?subject (subject))
     (?cc (cost-center))  (?name (name))))
  (instan411
     ((?subject (subject))
       (?cc (cost-center))  (?name (name)))) )
                 ---------------------------   join-list1
```

And if 'instan405' and 'instan455' are the form instance of form 'arvl' which satisfy the query:

```
        (?arvl
            (instance-of ARVL)
            (manager-signed Susan))
```

then the 'join-list' of query

```
        (?arvl  (instance-of ARVL)
                 (name ?name)
                (cost-centre ?cc)
                 (manager-signed Susan)
                (teaching-course
                   (subject ?subject)))
```

would be:

```
        ( (instan405
            ((?name (name))
             (?cc (cost-center))
              (?subject (teaching-course
                          (subject)))))
          (instan455
             ((?name (name))
             (?cc (cost-center))
          (?subject (teaching-course
                      (subject)))))) )
 ---------------    join-list2
```

To process a joined form reference pattern, two basic functions are needed. One is to join two elements of two join-lists, called **element-join,** the other is to join a join-list with a production-list, called **production-join.**

4.2.3.1 Element-Join Function

To join two elements of two join-lists is to judge if the join conditions between the two elements are satisfied. A join condition is defined if there are two variables which have the same variable name. A join condition is true if the values of both form instances that correspond to the variable are equal. The Formbase System has developed a function **(attri-refer instan attri-path)** which returns the instance's value of the attribute which is referred by the attribute path 'attri-path'. Therefore, if we join

```
(instan402  ((?subject (subject))
             (?cc (cost-center))
              (?name (name))))
```

and

```
(instan405
  ( (?name (name))
   (?cc (cost-center))
    (?subject (teaching-course (subject)))))
```

the join conditions are:

```
(and
 (equal  (attri-refer   'instan402   '(name))
            (attri-refer   'instan405   '(name)))
 (equal (attri-refer 'instan402 '(cost-center))
         (attri-refer 'instan405 '(cost-center)))
 (equal (attri-refer
            'instan402  '(time-tab (subject)))
        (attri-refer
          'instan405 '(teaching-course
                          (subject)))))
```

If two form instances do not have relational constraints between them, which means they do not have variables which have same names, then two of them always can be joined.

4.2.3.2 Production-Join

The result of joining join-lists together is a **production-list**. An element of a production-list is called a **production**. Each production contains the elements of each join-lists, and all of these elements can be joined with each other. For example, to join the above given join-list1 and join-list2, we get the following production-list:

```
(
 ( (instan402
     ((?subject (subject))
      (?cc (cost-center))
       (?name (name)))
   (instan405
     ((?name (name))
      (?cc (cost-center))
       (?subject (teaching-course (subject)))))
 )
 ( (instan411
     ((?subject (subject))
      (?cc (cost-center))
       (?name (name)))))
   (instan455
     ((?name (name))
      (?cc (cost-center))
       (?subject (teaching-course (subject))))))
 )
)
```

This production-list contains two productions:

```
( (instan402
    ((?subject (subject))
     (?cc (cost-center))
      (?name (name)))
   (instan405
     ((?name (name))
      (?cc (cost-center))
       (?subject (teaching-course (subject))))) )
```

and

```
( (instan411
    ((?subject (subject))
     (?cc (cost-center))
      (?name (name)))))
   (instan455
    ((?name (name))
     (?cc (cost-center))
      (?subject (teaching-course (subject))))))
```

This means that the form instances instan402 and instan405, instan411 and instan455 can be joined together according to current joining requirements.

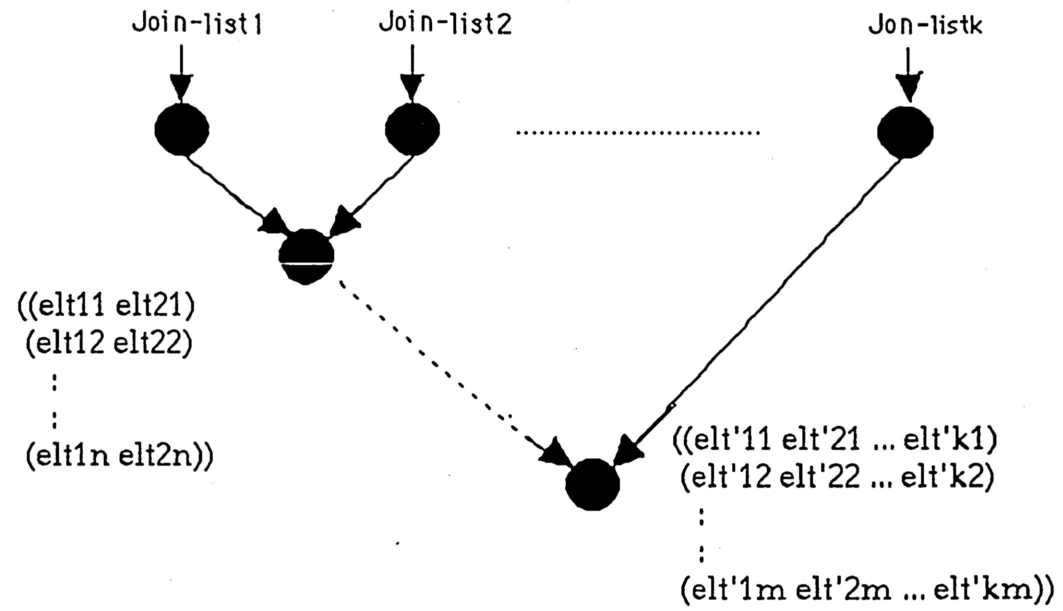

Join-list1
Join-list2
Jon-listk
((elt11 elt21)
(elt12 elt22)
:
:
(elt1n elt2n))
((elt'11 elt'21 ... elt'k1)
(elt'12 elt'22 ... elt'k2)
:
:
(elt'1m elt'2m ... elt'km))

Figure 4.8

To join more than one join-lists, the system first joins two of them, which generates a production-list as explained in above. It then joins the rest of the join-lists with this production-list. This process is shown in Figure 4.8. After the join of 'join-list1' and 'join-list2', the system generates a production-list as the following:

```
( (elt11  elt21)
  (elt12  elt22)
    :
    :
  (elt1n  elt2n) )
```

In this 'production-list', elt1j (j= 1, ... , n) belongs to Join-list1, and elt2j (j= 1, ... , n) belongs to Join-list2. Each production (elt1j elt2j) satisfies the join requirements. Then the system goes on to join the rest Join-lists with the generated production-list. With the progress of the joining process, the production-list and the productions inside it change their structures. For example, to join a Join-list3 with the generated production-list, for each element in Join-list3, the system checks if it can join with every elements in every production of the production-list. If it can join with every elements of a production, then it will be pushed into the production, this production will then be a new production for the new production-list of the join results. This process goes on and on, until the final production-list is generated as the following:

```
( (elt'11  elt'21  ... ...  elt'k1)
  (elt'12  elt'22  ... ...  elt'k2)
   :
   :
  (elt'1m  elt'2m  ... ...  elt'mk) )
```

In the implementation, this production-list is processed to return sets of form instances with their type information, therefore, a query for the all the instances of a certain type which matches the join constraints can be returned. This function is required by the form base operations such as 'set-value', 'delete-value' and 'modify-value'.

The ability to generate the production-list also provides a potential for developing a rule production system based on form reference patterns.

4.3. Conclusions

The development of the data definition and manipulation language for the formbase system enables us to manipulate nested data through the familiar office form concepts. It opens the potential to further develop Spread Sheet type of software.

Chapter 5.

Representation of Information Processing Activities upon Forms

Based on the developments of the Formbase System, this chapter discusses the representation of the information processing activities upon forms. The identity of an activity has been studied for more than twenty years. Started from the pioneer work of John McCarthy and Patrick Hayes [McCarthy, J. and Hayes, P., 1969], contributions have been made by Richard Fikes, Nils Nilsson, Drew McDermott, James Allen, Robert Moore, and Thomas Dean in [Fikes, R. and Nilsson, 1970; McDermott, D., 1982; Allen, J., 1984; Moore, R., 1985; Dean, T. and Dermott, D., 1987]. Currently there are two styles of approaches for representing an activity: one is time-slots based [Allen, J., 1984], the other is state-based [Fikes, R. and Nils Nilsson, 1970]. The time-slot based approach uses temporal logic [McDermott, D., 1982] and reasoning mechanism to reason about activities, while the state-based approach uses the add-list and delete-list representation [Fikes, R. and Nils Nilsson, 1970] and the heuristic searching to solve problems. The representation based on reasoning mechanism is logically sound and accurate, but the axiom and constraint rules that it needs for representing even a trivial activity are complex, the knowledge acquisition for an office information system would be enormous. The state-based activity representation suits knowledge acquisition better, but it needs the ability of sensing and switching background situation, at which today's computer systems are found to be poor. Because of these technical details, we do not have a marvellous choice. A practical system only can be implemented by properly "cutting corners" within an application domain. In the following, I first present a brief investigation of the developed activity representation methods, then develop the representation for the activities upon forms. This activity representation is the basis of the development of the Intelligent Formbase System (IFS).

5. 1. Review of Activity Representations and Problem Solving

Problem solving is a very common activity in normal life, but in computer science, no matter whether the approach is logical or heuristic, identifying a "neat" algorithm of problem solving for general purposes has proved to be very difficult. Currently there are three mechanisms which are clear and useful: one is a time-slot based activity representation whose problem solving process is based on reasoning mechanism; one is a state-based activity representation whose problem solving process is similar to the Modal Truth Criterion [Chapman, 1987]; the third is also state-based activity representation, but its problem solving process is based on reasoning resources of the activities. First of all, this review starts with an introduction to situation calculus and the frame problem.

5.1.1 Situation Calculus and Frame Problem

In 1969, John McCarthy and Patrick Hayes presented their work on representing activities [McCarthy, J. and Hayes, P., 1969]. They identified a set of concepts which are generally used by the researches that follow them.

- A *situation s* is the complete state of the universe at an instant of time. A system can never completely describe a situation; it can only give facts about the situations. These facts will be used to deduce further facts about that situation, about future situations and about situations that persons can bring about from that situation.

- A *fluent* is a function whose domain is the space of the set of all situations, denoted by *Sit* . If the range of the function is (true, false), then it is called a *propositional fluent.* If its range is *Sit* , then it is called a *situation fluent.* Fluents are often the values of functions. Thus raining(x) is a fluent such that raining(x) (s) is true if and only if it is raining at the place x in the situation s. This is also written as raining (x, s) making use of the equivalence between a function of two variables and a function of the first variable whose value is a function of the second variable.

- Causality: assumptions of causality can be made by means of fluent F(π) where π is itself a propositional fluent. F(π, s) asserts that the situation s will be followed (after an unspecified time) by a situation that satisfies the fluent π. We may use F to assert that if a person is out in the rain he will get wet, by writing:

$$\forall x.\ \forall p.\ \forall s.\ \text{raining}(x, s) \wedge \text{at}(p, x, s) \wedge \text{outside}(p, s) \supset F(\lambda s'.\ \text{wet}(p, s'), s)$$

- Actions: A fundamental role in study of actions is played by the situation fluent. For example, if *p* is a person, ¢ is an action or more generally a strategy, and *s* is a situation.

$$\text{result}\ (p, ¢, s)$$

denotes a situation that results when *p* carries out ¢, starting in the situation *s* . If the action or strategy does not terminate, result(p, ¢, s) is considered undefined. With the result fluent, a formula about a person who has ability to open a safe can be expressed in the following:

$$\text{has}\ (p, k, s) \wedge \text{fits}\ (k, sf) \wedge \text{at}\ (p, sf, s) \supset \text{open}\ (sf, \text{result}(p, \text{opens}(sf, k), s))$$

In this formula, *k* denote a key that fits the safe *sf* , then in the situation resulting from his performing the action opens(sf, k), that is, opening the safe *sf* with the key *k*, the safe is open.

The above concepts illustrate formalism of the initial attempt for modellingand reasoning activities. This formalism opened discussion of the frame problem. The frame problem is defined by Hayes as follows:

"given a certain description of a situation *s* ---- a collection of statements of the form f[s], where the brackets means that every situation in f is an occurrence of 's' ---- a system wants to be able to infer as much as possible about result(¢, s). Of course, what the system can infer will depend upon the properties of ¢. Thus the system requires assertions of the form:

$$f1[s]\ \&\ y(¢) \supset f2[\text{result}(¢, s)]$$

such an assertion is called a law of motion. The *frame problem* can be briefly stated as the problem of finding adequate collections of laws of motion."

The laws of motions are needed because logically *s* and result(¢, s) are different entities and there is no a priori justification for inferring any properties of result(¢, s) from those of *s*. However it is not possible for us to identify a sufficient set of these laws of motion by using the situation calculus, since the number of laws which are needed to describe a motion depend on the situation. Hayes gave an example in his paper for this problem.

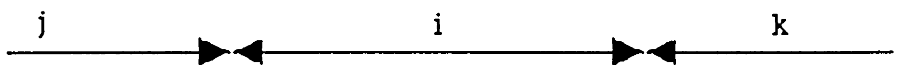
"Suppose I am describing to a child how to build towers of bricks. I say 'You can put the brick on top of this one onto some other one. if that one has not got anything else on it.' The child knows that the other blocks will stay during the move. But if I write the corresponding law of motion:

$$(\mathrm{on}\ (b1, b2, s)\ \&\ \forall z.\ \neg \mathrm{on}(z, b3, s)) \supset \mathrm{on}\ (b1, b3, \mathrm{result}(\ \mathrm{move}(b2, b3), s))$$

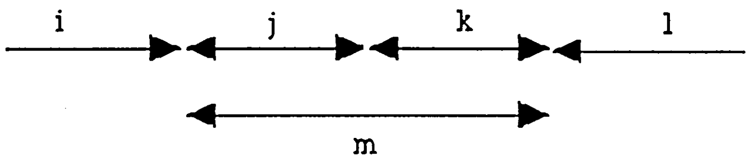
Then nothing follows concerning the other blocks. What assertions could we write down which would capture the knowledge that the child has about the world?"

5.1.2 Time-Slot Based Activity Representation

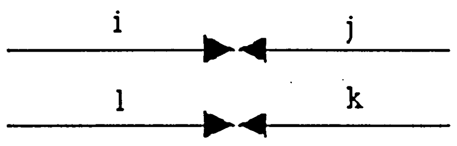
The time-slot based action representation and problem solving methods were developed through the efforts of Drew McDermott, James Allen, and Thomas Dean. A detailed introduction of this mechanism can be found in [Allen, J., et al, 1991]. This mechanism is a logical approach to action reasoning. The first order logic is augmented by temporal logic, then a set of axioms for activities and planning process are identified to support activity representation and the problem solving.



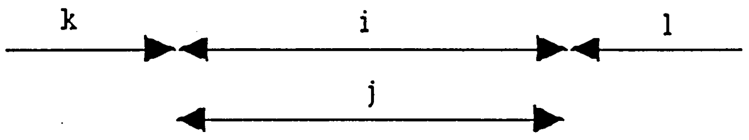
(1) For every period i, there is a j that meets it, and a k that is meets



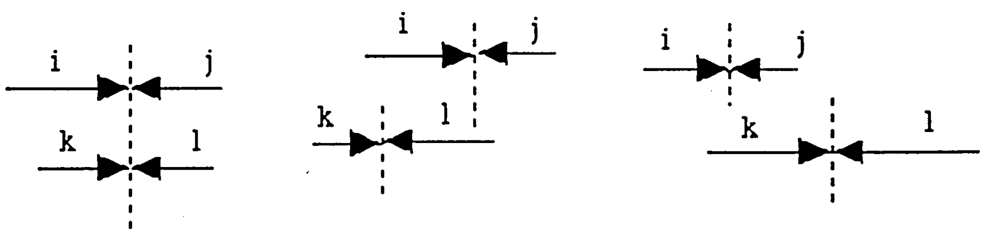
(2) For any j and k, where j meets k, there is a period m=j+k



(3) If i meets j and also meets k, and l that meets j also meets k



(4) If k meets i and j, and i and j meet l, then i=j



(5) The three ways that two pairs of meeting periods can be ordered

Figure 5.1

5.1.2.1 Primitives for Time Periods

The temporal logic is based on a simple set of primitives for time periods. A time period is the intuitive time associated with some event or property of the world. Intuitively, two time periods *m* and *n* meet if and only if *m* precedes *n*, yet there is no time between *m* and *n*, and *m* and *n* do not overlap. Therefore the axiomatization of the meets relation can be listed as follows, where *i, j, k, l,* and *m* are logical variables restricted to time periods. [Allen, J., et al, 1991], and they are shown as Figure 5.1.

(1) Every period has a period that meets it and another that it meets.

$\forall$ i, $\exists$ j,k Meet(j,i) $\wedge$ Meets(i,k).

(2) Periods can compose to produce a larger period. In particular, for any two periods that meet, there is another period that is the "concatenation" of them.

$\forall$ i,j,k,l. Meet(i,j) $\wedge$ Meets(j,k) $\wedge$ Meets (k,l)

$\supset \exists$. Meets(i,m) $\wedge$ Meets(m,l).

(3) Periods uniquely define an equivalence class of periods that meet them.

$\forall$ i,j,k,l. (Meets(i,j) $\wedge$ Meets (i,k) $\wedge$ Meets(l,j)) $\supset$ Meets(l,k).

(4) If two intervals both meet the same period, and another period meets both of them, the periods are equal:

$\forall$ i,j ($\exists$ k,l. Meets(k,i) $\wedge$ Meets (k,i) $\wedge$ Meets (i,l) $\wedge$ Meets(j,l)) $\supset$ i=j.

(5) The three ways that two pairs of meeting periods can be ordered.

$$\forall\, i,j,k,l.\ (\mathrm{Meets}(i,j) \wedge \mathrm{Meets}\ (k,l)) \supset (\otimes\ \mathrm{Meets}(i,l)$$
$$(\exists\, m.\ \mathrm{Meets}(k,m) \wedge \mathrm{Meets}(m,j))$$
$$(\exists\, m.\ \mathrm{Meets}(i,m) \wedge \mathrm{Meets}(m,l)))$$

With these axioms, one can define the complete range of the intuitive interval relationships that could hold between time periods. For example, the following formula describes that one period is before another if there exists another period that spans the time between them:

$\mathrm{Before}(i, j) \equiv \exists m.\ \mathrm{Meets}(i, m) \wedge \mathrm{Meets}(m, j).$

In this way, we could define Overlaps(i, j), Starts(i, j), During(i, j) and Finishes(i, j), and even define more relationships based these definitions such as:

$\mathrm{In}(i, j) \equiv \mathrm{Starts}(i, j) \vee \mathrm{During}(i, j) \vee \mathrm{Finishes}(i, j) \vee i=j$
$\mathrm{SameEnd}(i, j) \equiv \mathrm{Finishes}(i, j) \vee \mathrm{Finishes}(j, i) \vee i=j$

5.1.2.2 Properties of Temporal Logic

Then a temporal logic can be developed. The properties that are identified for temporal logic are Homogeneity, Concatenability and Countability. A proposition is homogenneous if and only if when it holds over a time period T, it also holds over any period within T. That is:

$\forall\, i,j.\ P(i1, \ldots, in, t) \wedge \mathrm{In}(t, t') \supset P(i1, \ldots, in, t')$

A proposition is called concatenable if and only if whenever it holds over time i and j, where i meets j, then it holds over the time i+j:

$\forall$ i,j. P(i1, ..., in, t) $\wedge$ P(i1, ..., in, t') $\wedge$ Meets(t, t') $\supset$ P(i1, ..., in, t+t')

A proposition is called countable if none of the times over which it holds overlap, that is, they are disjoint or equal:

$\forall$ i,j. P(i1, ..., in, t) $\wedge$ P(i1, ..., in, t') $\supset$ t (< m = mi >) t'.

5.1.2.3 Events Representation in Temporal Logic

Based on these temporal axioms, events can be represented. Every instance of an event defines uniquely the variables and the time over which it occurs:

$\forall$ e, i1, ..., in, t, i'1, ..., i'n, t' . E(i1, ..., in, e, t) $\wedge$ E(i'1, ..., i'n, e, t')
$\supset$ (i1=i'1 Ÿ , ..., Ÿ in = i'n Ÿ t = t').

For example to represent the event of stacking one object *a* to another object *b* , the following two axioms can be identified. The first axiom concerns the temporal structure of the activity, the second concerns the changes to situation that it brings about. In the axioms, the preconditions of the event have the prefix "pre", the effects of the event have the prefix "eff", and the conditions that must occur during the event have the prefix "con".

(1) Temporal Structure
$\forall$ e, $\exists$ a, b, e. Stack(a, b, e) $\supset$

$\text{Overlaps}(\text{pre1}(e), i) \wedge \text{Finishes}(\text{con1}(e), i) \wedge$

$\text{Meets}(\text{pre1}(e), \text{con1}(e)) \wedge \text{Meets}(i, \text{eff1}(e)) \wedge$

$\text{SameEnd}(i, \text{pre2}(e)) \wedge \text{Meets}(i, \text{eff2}(e))$

(2) Stacking Axiom1

$\forall e, \exists a, b, e.\ \text{Stack}(a, b, e) \supset$

$\text{Clear}(a, \text{pre1}(e)) \wedge \text{Holding}(a, \text{con1}(e)) \wedge \text{Clear}(a, \text{eff1}(e)) \wedge$

$\text{Clear}(b, \text{pre2}(e)) \wedge \text{On}(a, b, \text{eff2}(e))$

The above two axioms describe the knowledge which is independent of the situation. There exists however knowledge about actions which is relevant only in certain situations. For example, if the object *a* the system intends to move is on another object *c*, then after *a* is moved, *c* will be left clear. In order to describe such change, the following axiom should be added into the system.

$\forall i, a, b, c, t, e.\ \text{Stack}(a, b, e, i) \wedge \text{On}(a, c, t) \wedge \text{Overlaps}(t, i)$

$\supset \text{Clear}(c, \text{eff3}(e)) \wedge \text{Meets}(t, \text{eff3}(e)) \wedge \text{Meets}(t, \text{con1}(e)).$

5.1.2.4. A Summary

From the above, we can see how the temporal axioms are coming to the point to support activity representation. Since time-slot based mechanism takes a universal view of time and states, the frame problem does not exist. Therefore, once a system has the activity representations, an inference engine always can be implemented to support the problem solving for the activities. More discussion of the axioms concerning problem solving can be found in [Allen, J., et al, 1991].

However, since temporal logic is the foundation of time-slot based activity representation, the effects and the conditions of an event or an activity are described in a very "historical perspective", the actual problem solving methodologies at action point are not able to be represented or applied.

5.1.3 State-Based Activity Representation

In 1971, Richard Fikes and Nils Nilsson developed a state based activity representation in STRIPS planner [Fikes, R. and Nils Nilsson, 1971]. An activity in STRIPS is represented by preconditions, delete list and add list. For example, consider an activity push(k, m, n) for pushing object *k* from *m* to *n*. It can be described as follows:

precondition: ATR(m) $\wedge$ AT(k, m)

delete list: ATR(m); AT(k, m)

add list: ATR(n); AT(k, n)

It supposes that for all the *wffs* that are not metioned in the delete or add lists will remain the same. Compared to the situation calculus, this representation method provides a better modellingmeans for catching up knowledge of motion, since when people communicate to each other, they usually only describe the changes.

Modifying only the properties which have been changed is abstracted by STRIPS as a principle to deal with the frame problem, that is, a new state has the properties of add-list and inherits all the properties of the parent state unless they are mentioned in the delete-list. Therefore, different from time-slot based activity representation, the problem solving process for state-based activity representation maintains a state network. Its goal achieving process expands the network until a goal state has been acquired. Currently most of the developed goal achieving processes are an incomplete interpretation of MTC [Chapman, 1987].

In literature, many people emphasis the difference between STRIPS and the later developed planner such as NOAH [Sacerdoti, 1975], NONLIN [Tate, 1977] and SIPE [Wilkins, 1984], but the activity representation of these planners is similar to each other. Further developments were made by Saceroti [Sacerdoti, 1975] and Tate [Tate, A., 1977] by augmenting a goal structure which enable the activity representation to catch up the hierarchical structure of an activity, but the goal achieving process is still based on GPS [Newell, A., and Simon, H.A., 1970]. In the following, a more detailed discussion for the Modal Truth Criterion and the problem solving process based on reasoning resources are presented.

State-based activity representation suits to the application where the identities of states can be identified through a sequence of activities, and where the activity sequence is always generated before the execution.

5.1.3.1 Modal Truth Criterion

Solving problems by reducing differences is a general principle (GPS) which was identified by Newell and Simon in 1960's [Newell and Simon, 1970]. Based on GPS, Sacerdoti developed a heuristic searching process which is able to handle hierarchical planning problem by using a data structure called TOME (Table Of Multiple Effects) [Sacerdoti, 1975]. Austin Tate further developed the searching process by identifying the GOST (GOal STructure) structure [Tate, A., 1977]. The clarity and the applicability of the searching process come from backward chaining mechanism and predicate representation of the world. Because of predicate representation of the world, through unification process, the interactions between different activities can be assessed after every extension of the network, and therefore least-commitment of sub-goals is possible. This clarity was revealed by Chapman by his identification of the Modal Truth Criterion (MTC) for the assertions in the planning process [Chapman, 1987].

MTC: A proposition *p* is necessarily true in a situation *s* iff two conditions hold: there is a situation *t* equal or necessarily previous to *s* in which *p* is necessarily asserted; and for every step C possibly before *s* and every proposition *q* possibly codesignating with *p* which C denies, there is a step W necessarily between C and *s* which asserts *r*, a proposition such that *r* and *p* codesignate whenever *p* and *q* codesignate.

This criterion can be expressed by the following expression:

$$\begin{array}{l}
\exists t \Box t \leq s \wedge \Box \text{ assert-in}(p,\ t) \wedge \\
\quad \forall C \Box s \leq C \vee \\
\qquad \forall q \Box \neg \text{denies}(C,\ q) \vee \\
\qquad\quad \Box q \neq p \vee \\
\qquad\quad \exists W \Box C < W \wedge \\
\qquad\qquad \Box W < s \wedge \\
\qquad\qquad\; \exists r \text{ asserts}(W,\ r) \wedge p \approx q \Rightarrow q \approx r
\end{array}$$

It can be interpreted more clearly as a logical programming procedure. For the achievement of the precondition p of activity *R*, we first have a contributor which asserts *p* which occurs before *R* in the plan. If there is no such a contributor, the procedure must create one by instantiating an appropriate operator schema and installing it in the net. If there is already a contributor for *p*, but it is not yet ordered before *R*, then we can simply add the requirement ordering. After this we will have a guarantied contributor for the precondition p for *R*. Next, the procedure must check for activities that possibly delete the condition *p*. *p* is possibly deleted if there is an activity in the plan that could come after *C*, the contributor, and before *R*, the point of requirement. Not all activities are relevant : only those that explicitly mention *p* (or can codesignate with *p*) on their delete list can actually delete *p*. If no such deleters exists, then the truth of *p* for *R* is guarantied. If there is such a deleter, then it must be ordered outside of the range over which *p* is expected to be held. This deleter may be rendered harmless by ordering it before *C*, the point of contribution, or after the *R*, the point of requirement.

Most of the developed non-linear planner is an incomplete interpretation of MTC. Chapman [Chapman, 1987] has proved a theorem for the correctness and completeness for such planning process: If a planner, given a problem, terminates claiming a situation, the plan it produces does in fact solve the problem. If the planner returns signalling failure or does not halt, no solution exists.

5.1.3.2 Reasoning of Resources

Although MTC can be "neat" and useful under certain circumstances, it is however not able to deal with the situation in forward chaining manner. If we process the activity of turning on a light by using MTC, we suppose that the condition of light on is the button of the switch is down. Therefore, if the button has already been down, the light should have been on, otherwise there should be an activity which can achieve the state of button down. Therefore if the preconditions of the activity which can achieve the button down state are true, then the light should on, otherwise there should be activities (such as removing the barriers towards the button, or whatever) which can achieve the preconditions of the activity which can achieve the button down state. The process can go on and on.

The big assumption of MTC is that it supposes action starts after a complete planning process. This is not proper for those application domains where we can not plan exactly what to do unless we reach certain level of execution. For modellingan open information system, we do need a problem solver which can handle problems in an open and forward chaining manner, but to identify a goal achieving process for such a problem solver has proved to be even more difficult. Since in an open system, problem solving is distributive, the distributed issues such as task decomposition, communication, disperse viewpoints, and so on have to be considered. More detailed discussion for these issues can be found in papers of Distributed Artificial Intelligence [Allen and Gasser, 1990].

There is a sightly different goal achieving process which was developed by Wilkins in his planner SIPE [Wilkins, 1984] which is forward chaining. Its problem solving process reasons the resources to resolve the interactions between the sub-activities, which means the system has to know not only the sub-activities which are available for the goal, but also the resources which are needed by these activities. This requirement makes the planning process not very much different from MTC as we described in last section.

5.2. Why a New AI Problem Solver

In the above section, the developed mechanisms for representing activities have been reviewed. In this section the representation for the activities upon forms will be represented. Before the discussion, the initiatives for choosing an AI problem solving process for modellingan information system, and reasons why we need to develop a new activity representation for the activities upon forms are illustrated first.

5.2.1 The Difficulties of System Analyzing

The difficulties for analyzing a system is well known. So far we still do not have a method for organising a flexible as well as stable system in an open environment. Every methodology, no matter whether it is SSADM or SSM, tries to identify the structured aspects of a system. The analysis on the procedures of the system is required to be systematic and complete, and the operations upon data are required be clear before we actually store the data into a data base.

One benefit of using an AI problem solver to model an information system is that it can avoid the difficulties of analyzing the procedural knowledge of the system. As it is shown in Figure 5.2, the procedural knowledge acquisition needs not to be complete and systematic. For any procedural knowledge, as long as it could be defined in terms of goal, preconditions, effects, it could be captured and stored in the activity base of the system. The system can then support required task based on the procedural knowledge through a problem solving process.

However, as it has been pointed out, this architecture could not replace a system which needs more than one agent (problem solvers). For multi-user systems, further research is needed in Distributed Artificial Intelligence [Bond and Gasser, 1988].

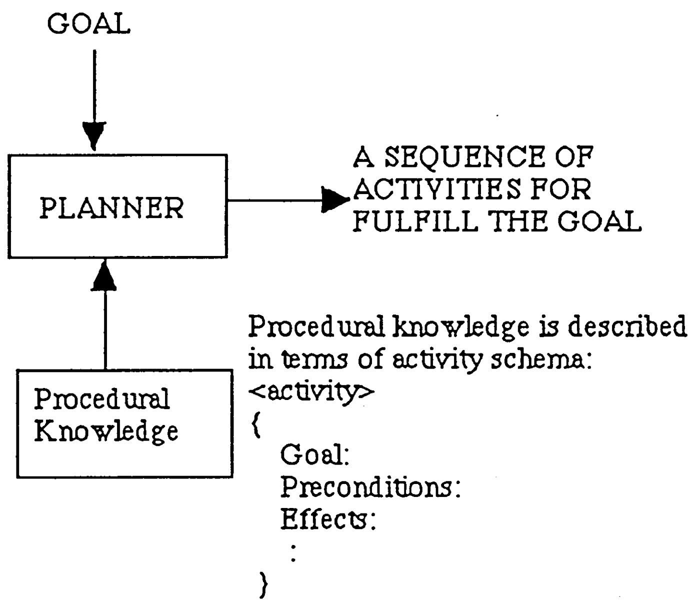

GOAL
PLANNER
A SEQUENCE OF ACTIVITIES FOR FULFILL THE GOAL
Procedural Knowledge
Procedural knowledge is described in terms of activity schema:
<activity>
{
Goal:
Preconditions:
Effects:
:
}

Figure 5.2

5.2.2 New Activity Representation ---- Situated Action

In AI action representations, actions are described in terms of their relation to situation descriptions, and the descriptions about situations are usually in first order predicates. But predicates are usually abbreviation of natural language rather than queries to the database system, therefore the action descriptions are not directly upon the database. In another words, the activities are not **situated**.

As we can see from the development of the Formbase System in Chapter 3 and Chapter 4, in the formbase, if we only use first order predicates in the task formalism to describe formbase activities, the description would not be situated in the formbase. The consequences of having activity descriptions not situated from the formbase are that we would not be able to catch up knowledge of form processing directly. This will bring about difficulties for modellingthe information systems.

5.3. Representation of the Information Activities upon Forms

5.3.1 The Requirements for Activity Representation in IFS

There are two basic requirements for the representation of the activities upon forms. First, as it has been explained above, the activity representation must be situated in the formbase. Second, the activities representation must be "nearly independent" to each other [Stefik, M. J., 1981]. This is because the knowledge of information processing of an organization is only partially well defined. Therefore the activity representation must provide a mechanism to identify the partially well defined procedural knowledge for processing forms in an office system.

5.3.2 Analysis for the Activities upon Forms

How is the procedural knowledge for processing forms partially organised? How should the activity representation be situated in the form base?

The key problem of representing an activity upon forms arises from the fact that the activity manipulates more than one type of information object. The task formalisms which are used by the developed AI planners, such as NONLIN and TWEAK, only use predicates, therefore can not effectively represent an activity which crosses different representation objects. For example, when a guest arrives a hotel, in order to register, the service desk needs to check if the guest has booked a room, if he has, then the room is assigned to him. This is a simple and ordinary activity, but it actually crosses different representational objects: the request for a room can be represented by a predicate, such as (room-request ?guest), but booking information is represented in a booking form. If we use a task formalism which only uses first order predicates to describe the activity, it turns out to be a superficial description rather than a well situated representation.

```
<room request>
{
  goal: (assign-room ?guest)
  precondition: (room-booked ?room ?guest)
  effects: (occupy ?guest ?room)
   :
}
```

In this representation, the predicates such as (room-booked ?x ?y) and (occupy ?x ?y), can not reveal the connections between the activity and the information objects (such as booking form). In order to improve this disadvantage, the solution is to enable the task formalism to use the Form Reference Patterns (see Chapter 3) for the information inside the formbase.

To augment the task formalism to refer to the Formbase, two aspects of activity representation need to be noticed. An activity representation has its effects and conditions which are **factual**, and its goal pattern and sub-goals which are **intentional**. For an activity upon formbase, the factual data are upon the formbase while the intentional data are used to communicate with other activity schemas. Therefore, when the task formalism is augmented to represent the information processing activities upon forms, we only need to augment the reference ability of its factual data. That is, the reference of the factual data should be directly upon the formbase by using the formbase query or formbase predicates (Chapter 3). The changes of the augmentation is shown by the Figure 5.3 and Figure 5.4. For traditional task formalism, goal pattern, conditions and effects are all represented by the first order predicates, but to represent the activity upon forms, within one activity representation we use both the first order predicates as well as the formbase query and formbase predicates.

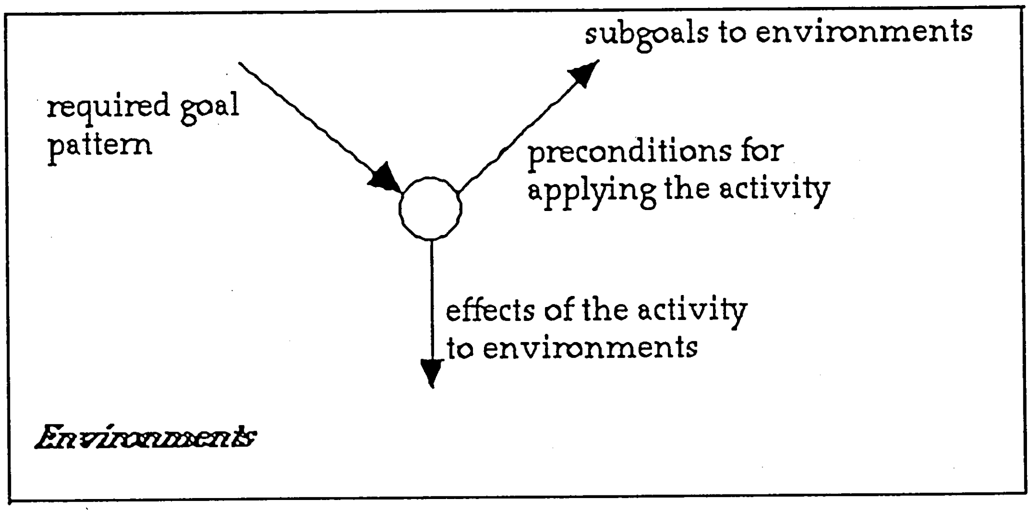

subgoals to environments
required goal pattern
preconditions for applying the activity
effects of the activity to environments
Environments

Figure 5.3

Base on the above discussion, representation for the activity upon form can be defined, it is called Formbase Activity Schema (FAS) in this thesis. Similar to state-base activity representation, slots such as goal-pattern, preconditions and effects are used. However the system draws a clear line between intentional data and factual data of formbase activity schemas. There are no the supervised preconditions similar to NONLIN system that can be interpreted as sub-goals. The sub-goal or assistant requirements of an activity is represented by a separated slot **:assistant-activities**. Instead of having supervised conditions, the formbase activity schema has a condition called **non-assistant-condition**. When **non-assistant-condition** is true, the **assistant-activities** slot of the activity schema will not be evaluated.

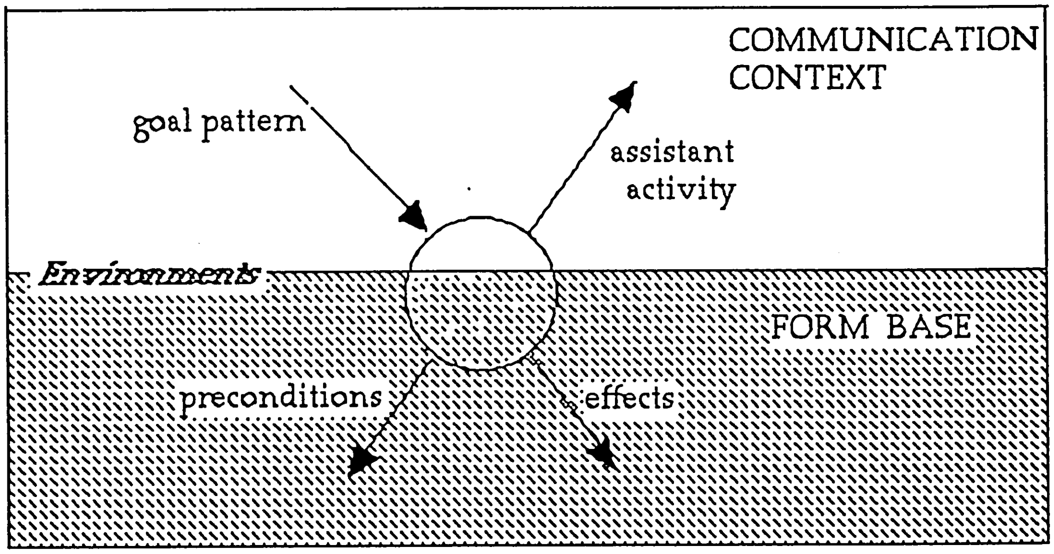


Figure 5.4

5.3.3 Representation for the Activity upon Forms

Therefore, an formbase activity schema upon form base has the following slots. Except 'goal-pattern', all the others could be optional for representing an activity.

```
<activity>
{
```

```
goal-pattern
holding-cond
non-assistant-cond
assistant-activity
effects
}
```

The semantic meaning of the slots are:

goal-pattern	: represents the aim of the activity in the first order predicate or relation patterns;
holding-cond	: conditions for applying activity schema, they are expressed in the formbase predicates;
non-assistant-cond	: conditions for not evaluating assistant-activities slot they are expressed in the formbase predicates;
assistant-activities	: assistant activities to achieve the goal pattern. If the non-assistant-cond is evaluated true, this slot will not be evaluated.
effects	: operations upon forms, they are represented by using formbase operations.

5.3.4 An Example

The following is an example of the activity schema. It describes the activity of checking the 'rvl' form (Figure 3.1) before payment. Basically it checks if the 'rvl' forms have been signed by the manager.

```
<check-rvl-payment>
{
   goal-pattern: '(check-rvl-payment $name)
   non-assistant-cond: '(every-equal
                         :forms  ((?rvl  (instance-of rvl)
                                         (name $name)
                                        (cost-centre ?cc))
                                 (?arvl  (instance-of arvl)
                                          (name $name)
                                         (cost-centre ?cc)))
```

```
                           :type rvl
                           :subarea-path (manager-sig)
                           :attribute manager-sig
                           :value Susan)
 assistant-activities: '(for (rvl1 :in
                               (formbase-query
                                 :forms
                                   ((?rvl
                                      (instance-of rvl)
                                      (manager-sig (manager-sig ~Susan))))
                             :do (sub-goals (manager-sig  rvl1)))
}
```

A formbase activity representation is basically written in Lisp code, besides the functions developed for formbase system in Chapter 3, there are two things that need more explanation. The first is the concept of **activity variable.** An activity variable, denoted as $x style, is used by an activity schema to represent the semantic links between the slots of the schema, its scope is within the activity schema, and it is interpreted within the schema. The reason why we need the activity variables is because an activity variable is different from the variable that is used by the form reference patterns, denoted in ?x style (see Chapter 3). The variables used by form reference patterns represent the links between different form types, and are interpreted by the query processing program which has been explained in Chapter 4. The activity variables are processed by the problem solving process which will be illustrated in Chapter 6.

The second thing that needs more explanation is the functions that are used by assistant-activity slot to seek other activities. There are two developed Lisp functions, namely **for** and **sub-goals**. Since a formbase query inside an activity schema may have many values, the **for** function provides an iteration structure to explore all the alternatives of the values. The **sub-goals** function pose the sub-goal pattern to the system as a goal pattern to search for assistant activities.

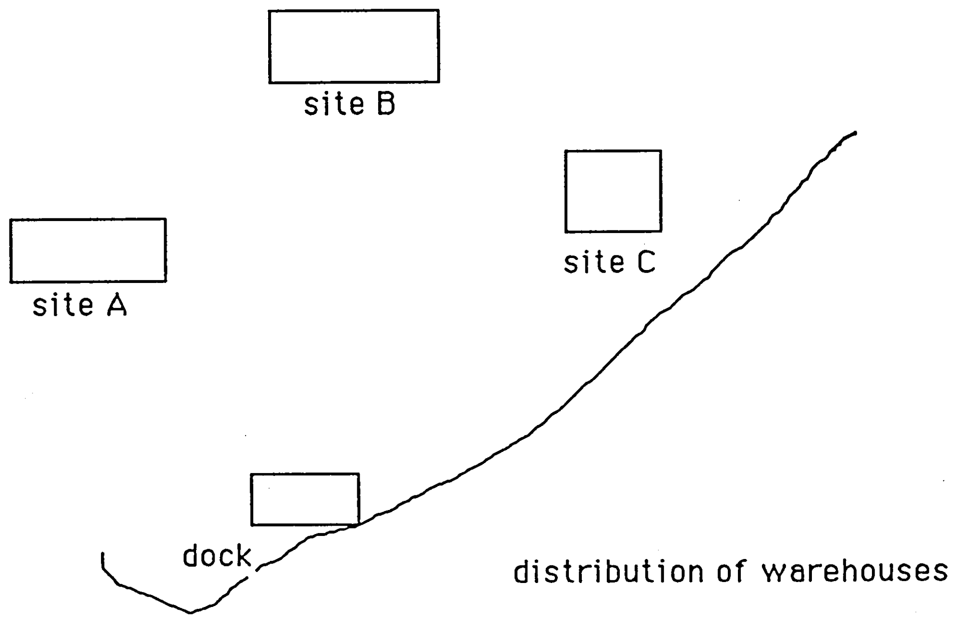


Figure 5.5

5.4. Modellinga Warehouse System

This section presents representations of the activities of a warehouse management system by using the activity schema concept developed in above section. The warehouses are shown in Figure 5.5. Four of them are located at site A, B, C and a dock. The storing and transfering management of them are based on four forms. They are the Table of Space (TS), the Table of Products (TP), the Table of Stored Products and the Goods Transfer Request Form (GTRF).

The TS table contains information of all the spaces of the warehouses (Figure 5.6), which includes the space identifier of a space, the status of whether a space has been occupied, and the size of a space. It is defined as follows:

```
(define-form
   (TS
      (site)
      (manager)
      (date)
      (association-of space-tab
           (space-id)
           (status)
           (space-size))))
```

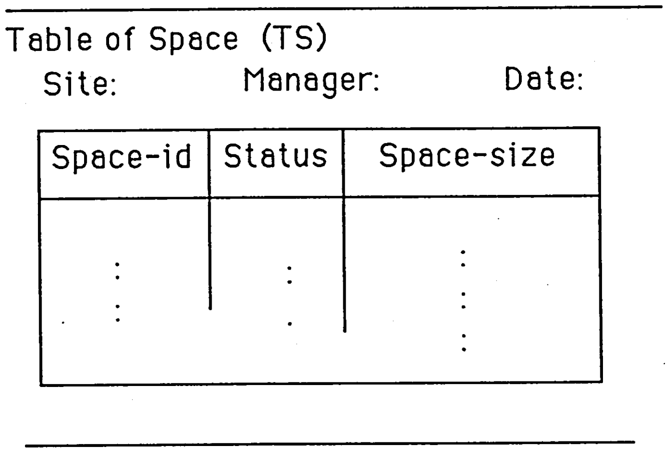


Table of Space (TS)

Site: Manager: Date:

Space-id	Status	Space-size
:	:	:
:	.	:
		:

Figure 5.6.

The TP table contains information of the products. which includes the type of a product and required space of it (Figure 5.7). It can be defined as follows:

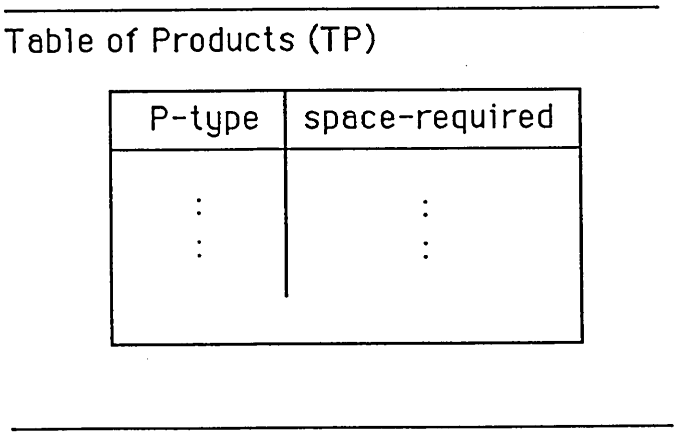


Table of Products (TP)

P-type	space-required
:	:
:	:

Figure 5.7

```
(define-form
  (TP
     (association-of  product-tab
          (P-type)
          (space-required))))
```

The TSP table is an actual notebook of a warehouse. It contains information of which product is stored at where (which space), and the product-type, entry-date, etc (Figure 5.8). It can be defined in the following:

```
(define-form
   (TSP
      (site)
      (manager)
      (total-space-left)
      (association-of  ware-house-tab
           (product-id)
           (product-type)
           (space-id)
           (entry-date))))
```

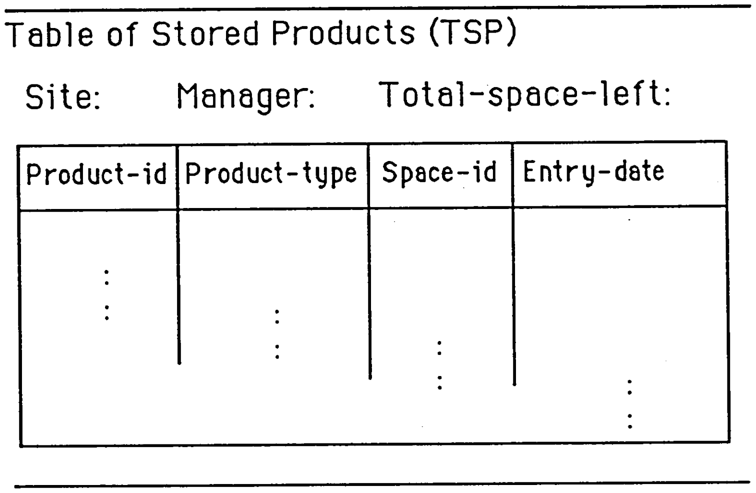


Table of Stored Products (TSP)

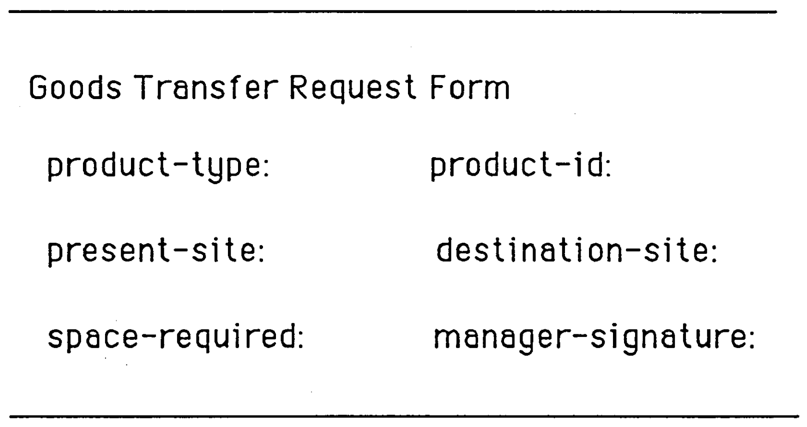
Site: Manager: Total-space-left:

Product-id	Product-type	Space-id	Entry-date
⋮	⋮	⋮	⋮

Figure 5.8

The Goods Transfer Request Form records request of transfering a product form one side to another side, it is shown as Figure 5.9, and can be defined as follows:

```
(define-form
   (GTRF
      (product-type)
      (product-id)
      (present-site)
      (destination-site)
      (space-required)
      (manager-signature)))
```


Goods Transfer Request Form

product-type:	product-id:
present-site:	destination-site:
space-required:	manager-signature:

Figure 5.9

Based on the above definition of forms, the operations of forms can be modelled. The first operation that we are going to represent is the transfer operation:

```
<transfer-product>
{
   goal-pattern: (transfer  $product-id  $departure  $destination)
   holding-cond:
        (exists
             ((?gtrf  (instance-of GTRF)
                      (product-id  $product-id)
                      (present-side  $departure)
                      (destination-side  $destination))))
   assistant-activities:
        (let ((pro-type (formbase-query
                          :forms ((?gtrf  (instance-of GTRF)
                                        (product-id $product-id)
                                        (present-side $departure)
                                         (destination-side $destination)))
                            :attribute product-type)))
                    (spa-size (formbase-query
                            :forms ((?gtrf (instance-of GTRF)
                                        (product-id $product-id)
                                        (present-side $departure)
                                         (destination-side $destination)))
                            :attribute space-required)))
                   (sub-goals (store $product-id pro-type spa-size $destination)))
        effects:
            (delete-subform
                 :forms ((?gtrf (instance-of GTRF)
                                (product-id $product-id)
                                (present-site $departure)
                                 (destination-side $destination))))
            (delete-subform
                 :forms ((?tsp (instance-of TSP)
                               (site $departure)
                               (ware-house-tab (product-id $product-id)))
                 :subarea-path (ware-house-tab) )
            (modify-value
                 :forms ((?ts (instance-of TS)
                              (site $departure)
                              (space-tab (space-id ?space-id)))
                        (?tsp (instance-of TSP)
                              (site $departure)
                              (ware-house-tab
                                    (product-id $product-id)
                                    (space-id ?space-id))))
                 :type TS
                 :subarea-path (space-tab)
                 :attribute Status
                 :value Free)
}
```

Whenever a product is required to transfer from one side to another, the 'holding-condition' is that there is a corresponding Goods Transfer Request Form, and the sub-goal is that the product can be stored at the destination site. As to the effects of the activity, the space occupied by the product at the departure site should be released and the transfer request which is recorded in the GTRF form should be removed.

The following is the representation of the activity of storing an activity into a ware-house.

```
<store-product>
{
   goal-pattern: (store  $product-id  $product-type $space-required $site)
   non-assistant-cond:
         (some-bigger
               :forms  ((?ts (instance-of TS)
                             (site $site)
                             (space-tab (status Free))))
               :subarea-path (space-tab)
               :attribute space-size
               :value $space-required)
   assistant-activities: (sub-goals (find-space $space-require $site)))
   effects:
         (let ((sp-id (one-of (formbase-query
                              :forms ((?ts (instance-of TS)
                                           (site $site)
                                           (space-tab
                                               (status Free)
                                               (space-size >=$space-required))))
                              :subarea-path (space-tab)
                              :attribute space-id))))
           (set-value :form ((?ts (instance-of TS)
                                  (site $site)
                                  (space-tab (space-id sp-id))))
                      :subarea-path (space-tab)
                      :attribute status
                      :value Free)
         (add-subform
                   :forms ((?tsp (instance-of  TSP)
                                 (site  $site)))
                   :subarea-path (ware-house-tab)
                   :value ($product-id $product-type pt-id (date))))
}
```

This activity first checks if there are spaces which are free and bigger than the required space, if there are such spaces, the Lisp code in effects parts are evaluated, otherwise the assistant activities will be evaluated. The assistance activity tries to find spaces for the product, the effects of having a product

The following is the representation of the activity of storing an activity into a ware-house.

```
<store-product>
{
   goal-pattern: (store  $product-id  $product-type $space-required $site)
   non-assistant-cond:
         (some-bigger
               :forms  ((?ts (instance-of TS)
                             (site $site)
                             (space-tab (status Free))))
               :subarea-path (space-tab)
               :attribute space-size
               :value $space-required)
   assistant-activities: (sub-goals (find-space $space-require $site)))
   effects:
         (let ((sp-id (one-of (formbase-query
                                 :forms ((?ts (instance-of TS)
                                              (site $site)
                                              (space-tab
                                                  (status Free)
                                                  (space-size >=$space-required))))
                                 :subarea-path (space-tab)
                                 :attribute space-id))))
           (set-value :form ((?ts (instance-of TS)
                                  (site $site)
                                  (space-tab (space-id sp-id))))
                      :subarea-path (space-tab)
                      :attribute status
                      :value Free)
         (add-subform
                    :forms ((?tsp (instance-of  TSP)
                                  (site  $site)))
                    :subarea-path (ware-house-tab)
                    :value ($product-id $product-type pt-id (date))))
 }
```

This activity first checks if there are spaces which are free and bigger than the required space, if there are such spaces, the Lisp code in effects parts are evaluated, otherwise the assistant activities will be evaluated. The assistance activity tries to find spaces for the product, the effects of having a product stored in the ware-house is that space is occupied, and the storing information has to be recorded in TSP form.

The following is the representation of the activity of finding a space.

```
<find-space>
{
  goal-pattern: (find-space $space-required $site)
  holding-cond:
      (some-bigger
           :forms1 ((?ts (instance-of TS)
                         (site $site)
                        (space-tab (status Occupied)
                                   (space-size >=$space-required))))
           :subarea-path1 (space-tab)
           :attribute1 space-size
           :forms2 ((?ts (instance-of TS)
                          (site $site)
                         (space-tab (status Occupied)
                                    (space-size >=$space-required)
                                    (space-id ?space-id)))
                    (?tsp (instance-of TSP)
                           (site $site)
                           (ware-house-tab
                                 (space-id ?space-id)
                                 (product-type ?product-type)))
                    (?tp (instance-of TP)
                          (product-tab (P-type ?product-type))))
           :type2 TP
           :subarea-path2 (product-tab)
           :attribute2 space-required )
  assistant-activity:
      (for (pro-id :in (formbase-query
                         :forms ((?tsp (instance-of TSP)
                                        (site $site)
                                        (ware-house-tab
                                              (space-id ?space-id)))
                                 (?ts (instance-of TS)
                                       (site $site)
                                      (space-tab
                                         (space-size >=$space-required)
                                         (space-id ?space-id))))
                         :type TSP
                         :subarea-path (ware-house-tab)
                         :attribute product-id))
```

```
        :do (if (bigger
                    :forms1 ((?ts (instance-of TS)
                                  (site $site)
                                  (space-tab (space-id ?space-id)))
                             (?tsp (instance-of TSP)
                                   (site $site)
                                   (ware-house-tab (space-id ?space-id)
                                                   (product-id pro-id))))
                    :type1 TS
                    :subarea-path1 (space-tab)
                    :attribute1 space-size
                    :forms2 ((?tsp (instance-of TSP)
                                   (site $site)
                                   (ware-house-tab (producet-id pro-id)
                                                   (product-type ?pro-type)))
                             (?tp (instance-of TP)
                                  (product-tab (P-type ?pro-type))))
                    :subarea-path2 (product-tab)
                    :attribute2 space-required)
                (sub-goals (move pro-id $site))))
}
```

Find-space activity first checks if there are products which have occupied a space that is bigger than the required space, and at the meanwhile is bigger than the need of the product itself. In this activity representation, to find out if the product has occupied a space which is bigger than necessary, three forms are used in the query. If there are products which have occupied a space that is bigger than the required space and is bigger than its need, then the system tries to move it. The following is the representation for the activity of moving a product. There are two situations for moving a product, in the first case, we re-arrange (move) a product inside the warehouse if a space can be found inside the warehouse. Secondly, if there is no free space in the warehouse, we check if there is a product which will be transfer out. In the following, we consider moving inside a warehouse first:

```
<moving-product-1>
{
```

```
goal-pattern: (move $product-id $site)
non-assistant-cond:  ;; check if there is free space which is bigger than
                     ;; the space that the product requires.
     (some-bigger
          :form1 ((?ts (instance-of TS)
                       (site $site)
                       (space-tab (status Free)))
          :subarea-path1 (space-tab)
          :attribute1 space-size
          :form2 ((?tsp (instance-of TSP)
                        (site $site)
                        (ware-house-tab (product-id $product-id)
                                        (product-type ?pro-type)))
                  (?tp (instance-of TP)
                       (product-tab (P-type ?pro-type))))
          :type2  TP
          :subarea-path2 (product-tab)
          :attribute2 space-required )
:assistant-activities
     (let ((sp-size (formbase-query
                      :forms ((?tp (instance-of TP)
                                   (product-tab (P-type ?pro-type)))
                              (?tsp (instance-of TSP)
                                    (ware-house-tab
                                     (product-id $product-id)
                                     (product-type ?pro-type))))
                      :type TP
                      :subarea-path (product-tab)
                      :attribute space-required)))
      (sub-goals (find-space sp-size $site)))
:effects
    (let* ((old-sp (formbase-query  ;; space the product stored
                      :forms ((?tsp (instance-of TSP)
                                    (site $site)
                                    (ware-house-tab
                                     (product-id $product-id))))
                      :subarea-path (ware--house-tab)
                      :attribute space-id))
          (req-sp (formbase-query  ;; the space-size the product required
                      :form ((?tsp (instance-of TSP)
                                   (site $site)
                                   (ware-house-tab
                                     (product-id $product-id)
                                     (product-type ?pro-type)))
                             (?tp (instance-of TP)
                                  (product-tab (P-type ?pro-type))))
                      :type  TP
                      :subarea-path (product-tab)
```

```
                    :attribute space-required )
          (new-sp (one-of    ;; space the product will move to
                    (formbase-query
                      :forms  ((?ts (instance-of TS)
                                    (site $site)
                                    (space-tab (status Free)
                                               (space-size >=rep-sp))))
                      :subarea-path (space-tab)
                      :attribute space-id)))
          (pro-type (formbase-query ;; find out the type of the product
                         :form ((?tsp (instance-of TSP)
                                      (site $site)
                                      (ware-house-tab
                                         (product-id $product-id))))
                         :subarea-path (ware-house-tab)
                         :attribute product-type)))
      (modify-value  ;; free the old space
          :forms ((?ts (instance-of TS)
                       (site $site)
                       (space-tab (space-id old-sp))))
          :subarea-path (space-tab)
          :attribute status
          :value  Free)
      (set-value  ;; record the occupation of the new space
          :forms ((?ts (instance-of TS)
                       (site $site)
                       (space-tab (space-id new-sp))))
          :subarea-path (space-tab)
          :attribute status
          :value Occupied)
      (delete-subform ;; delete the old space information
          :forms ((?tsp (instance-of TSP)
                        (site $site)
                        (ware-house-tab (space-id old-sp))))
          :subarea-path (ware-house-tab))
      (add-subform  ;; record the new occupation information
          :forms ((?tsp (instance-of TSP)
                        (site $site)
                        (ware-house-tab (space-id new-sp))))
          :subarea-path (ware-house-tab)
          :value ($product-id pro-type new-sp (date)) ))

}
```

Although the code of this activity is lengthy, the logic structure of this activity is simple, if there is a space which is free and bigger enough for the product, then we move the product into the space, otherwise we try to find a space for the product. The effects of moving a product requires a bit more code, since it means the space which was occupied by the product should be released after the moving, and the new space should be occupied. If there is no such spaces inside the ware-house, the following moving activity checks if there is a product will be transferred out.

```
<move-product-2>
{
   goal-pattern: (move-product $product-id $site)
   holding-cond:
        (some-bigger
             :forms1 ((?gtrf (instance-of GTRF)
                              (present-site $site)
                             (product-id ?pro-id))
                      (?tsp (instance-of TSP)
                             (site $site)
                             (ware-house-tab (product-id ?pro-id)
                                             (space-id ?sp-id)))
                      (?ts (instance-of TS)
                           (site $site)
                           (space-tab (space-id ?sp-id))))
             :type1 TS
             :subarea-path1 (space-tab)
             :attribute1 space-size
             :forms2 ((?tp (instance-of TP)
                           (product-tab (P-type ?pro-type)))
                      (?tsp (instance-of TSP)
                             (site $site)
                             (ware-house-tab (product-id $product-id)
                                             (product-type ?pro-type))))
             :type2 TP
             :subarea-path2 (product-tab)
             :attribute2 space-required )
   assistant-activities:
        (let ((rep-sp (formbase-query  ;;space required by the product
                      :forms ((?tp (instance-of TP)
                                   (product-tab (P-type ?pro-type)))
                              (?tsp (instance-of TSP)
                                    (site $site)
                                    (ware-house-tab
                                         (product-id $product-id)
                                         (product-type ?pro-type))))
```

```
                    :type TP
                    :subarea-path (product-tab)
                    :attribute space-required )))
      (for (mp-id :in (formbase-query
                          :forms ((?gtrf (instance-of GTRF)
                                         (present-site $site)
                                         (product-id ?pro-id))
                                  (?tsp (instance-of TSP)
                                        (site $site)
                                        (ware-house-tab
                                         (product-id ?pro-id)
                                         (space-id ?sp-id)))
                                  (?ts (instance-of TS)
                                       (site $site)
                                       (space-tab (space-id ?sp-id)
                                                  (space-size >=rep-sp))))
                          :type GTRF
                          :attribute product-id )
       :do (let ((dep (formbase-query
                          :forms ((?gtrf (instance-of GTRF)
                                         (present-site $site)
                                         (product-id mp-id)))
                          :attribute present-side))
                 (des (formbase-query
                          :forms ((?gtrf (instance-of GTRF)
                                         (present-site $site)
                                         (product-id mp-id)))
                          :attribute destination-side)))
             (sub-goals (transfer mp-id dep des)))))

}
```

The logic of this activity is also very simple, once there is a product which has occupied a space that we are looking for and is requested to be transferred to another side, the system first carry out the transfering process.

5. 5. A Summary

In this Chapter, the developed methods for representing activities have been reviewed, based on the investigation, a representation mechanism for the activities upon forms are developed, and an intensive example of how to use this mechanism to model a practical system is introduced. The problem solving algorithm for the activities is presented in the next Chapter.

Chapter 6.

A Problem Solving Process for the Activities upon Forms

This chapter presents a problem solving process to assist human information processing activities upon the formbase. The concept of formbase activity schema was been introduced in Chapter 5. The development in this chapter is based on the formbase activity schema and the concepts of the developed nonlinear planners, such as NOAH [Sacerdoti, 1975], NONLIN [Tate, A., 1977] and TWEAK [Chapman, 1987]. The problem solving process of nonlinear planners maintains a planning network/procedure network ---- a network of situations which are connected by activities ---- so that the requirements and the effects of every involved activity can be evaluated for the purpose of achieving a required goal. The issue of developing a problem solving process upon the formbase involves the following problems:

- how to represent the planning network;
- how to take account of effects of the activities;
- how to resolve the interactions between the subgoals.

In the following, the framework of the Intelligent Form System (IFS) is presented first, then the special nature of the problem solving for the activities upon forms is discussed. Finally a problem solving process for the Formbase activities is developed.

6.1. The Frame of the Intelligent Form System

The reason for developing a Intelligent Form System (IFS) is to improve the functionality, flexibility and stability of the formbased Office Information System. This requires certain flexibilities of the mapping between the task representation, the activity representation and the data representation. By using a problem solving process, the IFS is able to achieve such flexibility. The system frame of IFS is shown in Figure 6.1.

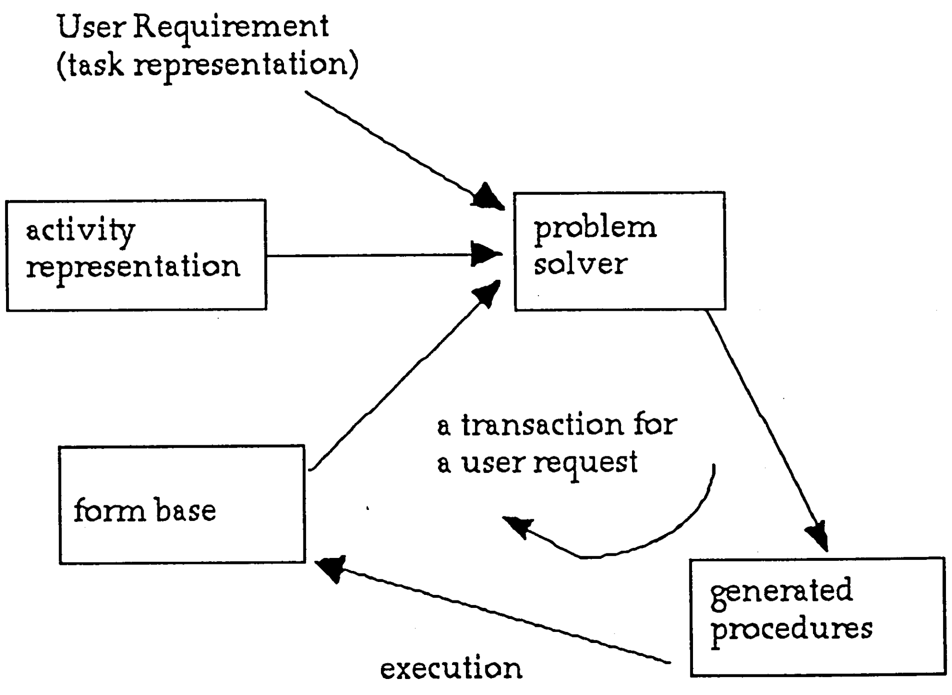


Figure 6.1

When user's requirement is input to the system, the problem solver works to generate a sequence of activities which can process the formbase so that the goal can be achieved. Since in the formbase, the activity representation and the task representation are independent of each other, and only relate to each other dynamically during the problem solving process, certain flexibilities of the system can be achieved.

6.2. The Special Nature of Problem Solving for the Activities upon Forms

This section discusses the special natures of the problem solving for the activities upon the forms. The most important feature of the activities upon forms is that they satisfy **Organizational Activity Assumption** which will be introduced later. This feature makes the problem solving upon forms different from previous problem solvers. In the following the principle of the developed state-based problem solvers is reviewed first, then a set of concepts for identifying the features of the activities upon forms are presented. Most of the developed problem solver deals with the activities which are performed by human-beings spontaneously. This thesis refers to them as **unstructured activities.**

6.2.1 The Principle of State-Based Problem Solving for Unstructured Activities

Problem solving is to identify a sequence of activities to fulfil a task from an initial state. Every state-based activity representation has a list of preconditions and a list of effects. The problem of choosing which activity and when to apply it to the system is systematically related to every aspect of the problem solving process. On one hand, the applicability of an activity depends on the situation before its application; on the other hand, every activity may modify the situation description. Therefore unless the order of the activities has been specified, the situation descriptions inside the planning network are nondeterministic. Thus a problem solver can not deduce its solutions based on the concept of situation.

The problem solving of a state-based planner is to take account of each predicate in the precondition list and effect list of every needed activity in terms of their logic dependency in the planning network. In other words, in the planning network, the truth of a predicate at a node can be judged based on both the initial situation and the effects of the activities. One important functionality of the problem solver is to maintain and to generate data for the logical connections and protections of each predicate. The main data structures for this mechanism are planning network/procedure network, Table Of Multiple Effects (TOME) [Sacerdoti, 1975], and the GOal STructure (GOST) [Tate,A., 1977]. Among them the planning network records the time order or the time relation of the activities, TOME records all the effects of the activities at every node, and GOST records the contributors of every condition.

Almost all the developed planners concern the daily activities which are performed automatically by human-beings. For daily activities, there is no commonly accepted symbolic information which can be used for defining the order of the activities, but why an activity is jeopardised by the others is usually clear. So inside a computer problem solver, the order can be generated by taking account of effects and requirements of every activity in the planning network. It is a main task of the planning systems to find out a right order of activity sequence to achieve the goal.

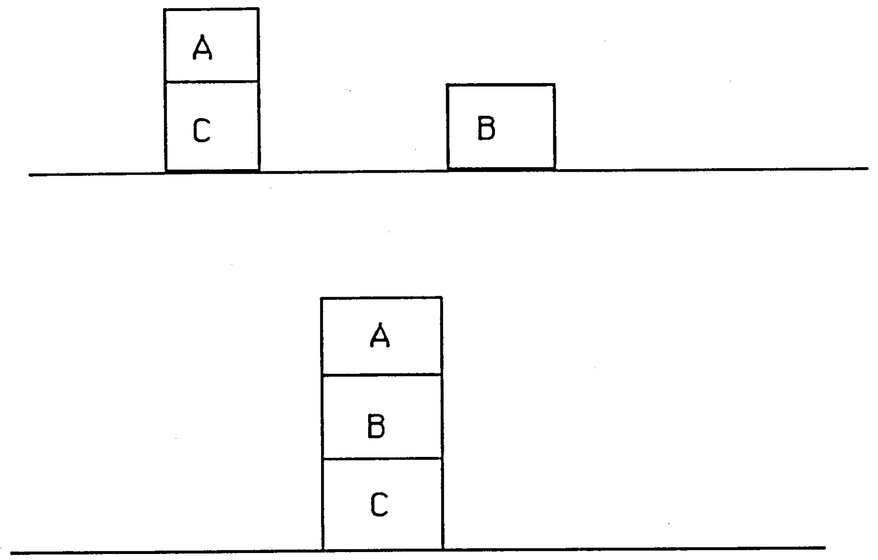


Figure 6.2

Figure 6.2 gives a very simple example of the unstructured activities. The initial situation is that block A is on block C, block B and C are on the table, and the top of block A and block B are clear. The goal is to build a tower of the blocks where block A is on block B, block B is on block C, and block C is on the table. A human will automatically first put block A on table, then put block B on block C, then put block A on block B to fulfil the task. The trouble is why he does not put block A on block B first, or put block B on block A first are so much related to the situation that it can not be abstractly represented. The basic knowledge about the process that can be represented independent to situation is the following activity:

```
<move>
{
  preconditions: (clear ?x) (clear ?y)
  add-list: (on ?x ?y)
  delete-list: (clear ?y)
}
```

When this activity representation is used by a problem solver to find the solution for the above question, efforts have to be made to check out how one move of a block will change the situation and how the moves are related to each other. The data structures such as TOME [Sacerdoti, 1975] and GOST [Tate, A., 1977] are identified for the use of resolve the interactions between the activities.

6.2.2 The Information Processing Activities upon Forms

The information processing activities upon forms, called *formbase activities,* have a different background. Forms and the formbase activities are consciously designed by an organization. Every formbase activity processes a certain set of data in the formbase, therefore there are no direct interactions between the formbase activities like the interactions between the activities in the block example. Whether a formbase activity is applicable depends on whether there is a set of data in the formbase which match the preconditions of the activity. Since every formbase activity may modify values of attributes of certain set of form instances, the actual scope of the activity ---- the instances which are influenced by the activity ---- is not clear unless the activity sequence is defined. The problem solving for formbase activities is therefore to identify the assistant activities needed and to determine the actual scopes of the effects of these activities.

6.2.2.1 Warehouse Example

Suppose we have two warehouses: one is on site A, the other is on site B, and each of them has five spaces, shown as in Figure 6.3. The *s1, s2, ..., s5* are space identifiers.

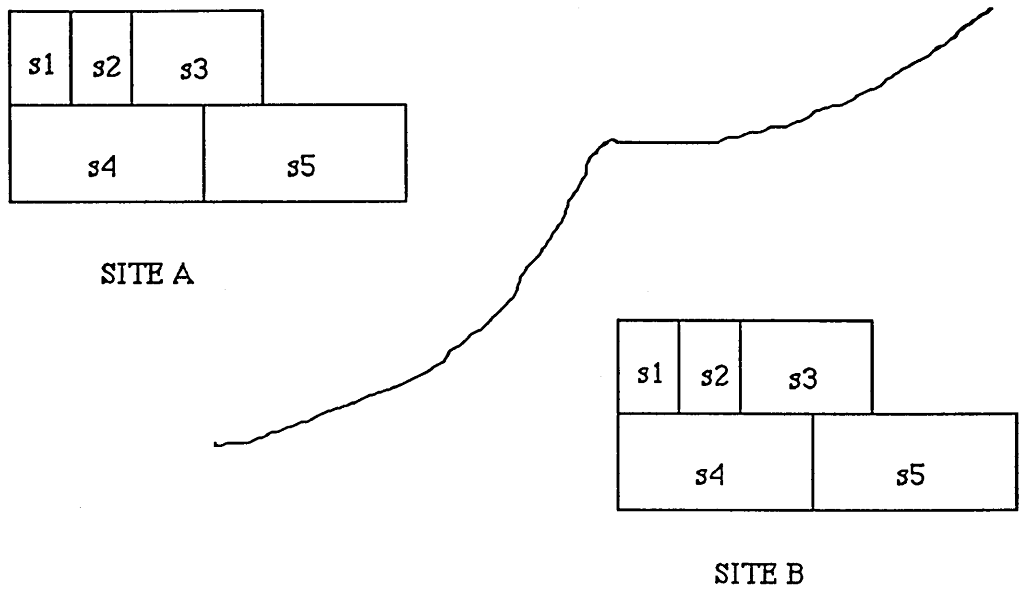


Figure 6.3

Suppose we only have three types of products which requires space size 1, 2, 3 respectively (Figure 6.4), and suppose that space size 3 is bigger than size 2, and size 2 is bigger than size 1.

Table of Products (TP)

P-type	space-required
pt1	1
pt2	2
pt3	3

Figure 6.4

Suppose we have seven products: *p1, p2, ..., p7,* and they are stored in the warehouses of the two sites as it is shown in Figure 6.5. The name *p1, p2, ..., p7* are product identifications.

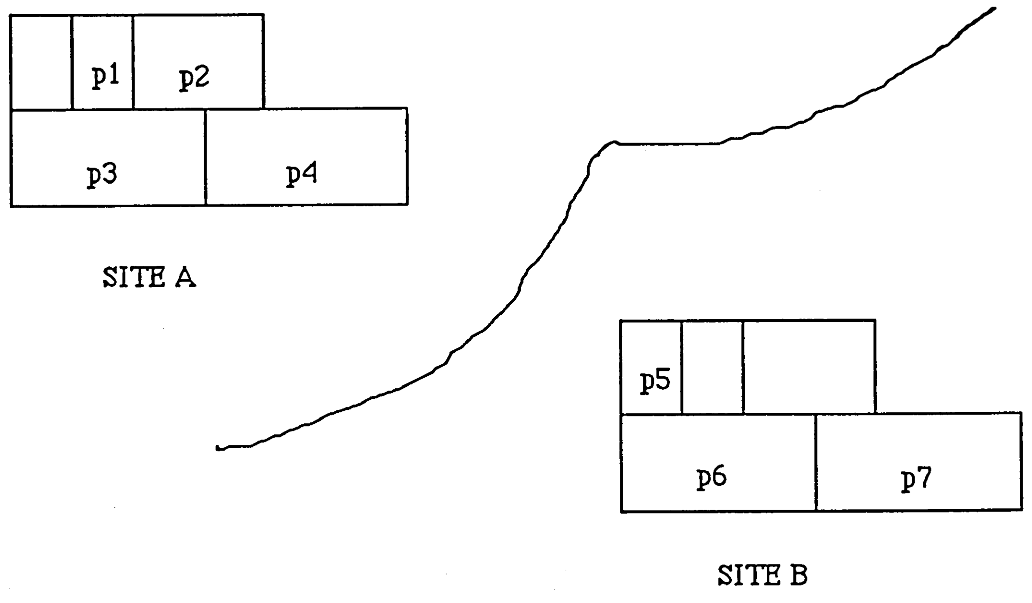


Figure 6.5

Therefore, we have the following two form instances (Figure 6.6) of the Table of Space (TS) form to describe the space situation of the two site. The *s1* space in site *A* is free, whose size is 1. The *s2* and *s3* spaces in site *B* are free, whose size are respectively 1 and 2.

The actual distribution of the products in the two warehouses are recorded in another two form instances of the Table of Stored Products (TSP) form (Figure 6.7). It can be seen that at Site *A*, space *s3* which is size 2 is occupied by *p2* which is type *pt1* and only needs size 1 space. Space *s4* and *s5* are occupied by *p3* and *p4* which are *pt2* type and only need size 2 space.

Table of Space (TS)

Site: A Manager: Ross Date:

Space-id	Status	Space-size
s1	Free	1
s2	Occupied	1
s3	Occupied	2
s4	Occupied	3
s5	Occupied	3

Table of Space (TS)

Site: B Manager: Bob Date:

Space-id	Status	Space-size
s1	Occupied	1
s2	Free	1
s3	Free	2
s4	Occupied	3
s5	Occupied	3

Figure 6.6

6.2.2.2 Organizational Activity Assumption

Now, suppose we want to transfer product p6 from site B to site A (Figure 6.5), let us check what will happen to the corresponding forms. Intuitively we know *p6* is of type *pt3* and needs a size 3 space, but all the size 3 spaces in site A have already been occupied. Fortunately the products which currently occupy the size 3 spaces *s4* and *s5* can be put into a size 2 space. Therefore we try to find an empty size 2 in the warehouse. The only size 2 space *s3* is occupied by another product *p1*, but this *p1* which is of *pt1* type and only needs a size 1 space. So, it can be moved to space *s1* which is free at this moment, therefore product p6 can be transfered from site *B* to site *A*.

Table of Stored Products (TSP)

Site: A Manager: Ross Total-space-left:

Product-id	Product-type	Space-id	Entry-date
p1	pt1	s2	
p2	pt1	s3	
p3	pt2	s4	
p4	pt2	s5	

Table of Stored Products (TSP)

Site: B Manager: Bob Total-space-left:

Product-id	Product-type	Space-id	Entry-date
p5	pt1	s1	
p6	pt3	s4	
p7	pt3	s5	

Figure 6.7

The computing modellingof the activities that are involved in the management of the warehouses such as 'find-space', 'move' and 'store' will manipulate the forms such as TS (Figure 6.6) and TSP (Figure 6.7). These activities have been discussed in Chapter 5. The physical appearance of these activities do not look different from the activities that are involved in the block example, but there is a profound difference between them. In the block example, an activity may manipulate an atomic entity more than once to achieve the goal, and an activity may directly override the effects of another activity. For example, in the block example, block A is first put on the table, then block B is put on block C, then block A is put on block B. During this process the block A is processed by the same activity twice for achieving the goal. Moreover, the goal is a set of predicates { (on A B), (on B C), (on C table), (clear A) }, they are achieved by different activities in the sequence of activities. There are possibilities that the required goal predicates are achieved by some activities, and later are overridden by some others, and then are asserted again.

In formbase activities, this does not happen. For example, in the warehouse example, no datum in the forms will be processed more than once by the same activity, and usually one activity will not directly override the effects of another. This is because the forms about the spaces and the products in the warehouses, and the operations upon them are consciously designed for the management of the warehouses. **An activity will not override the effects of the activity which searches it out.** This is generally true for all the information processing activities upon forms. Therefore it is abstracted as an assumption called **Organizational Activity Assumption.** The problem solver which is developed later in this chapter is based this assumption.

6.2.2.3 Interactions between Activities upon the Forms

Formbase activities obey the Organizational Activity Assumption, therefore there is no direct conflict between two activities which are designed to assist one another. The problem solving process will be simplified in terms of this, but there is another type of uncertainty and interaction.

In the IFS, since the FRPs can not unify each other (see Chapter 3), the problem solver upon forms can not use least-commitment methods (the method of using predicate patterns to submit sub-goals to the system) to process sub-goals/assistant-activities as those developed in problem solvers such as NONLIN [Tate, 1977] and MOLGEN [Stefik, 1981]. In the Formbase Activity Schema, the goal patterns for searching assistant activities are generated by using function **For** to explore every instance in the sets which are referred to by the FRPs. Therefore, every formbased activity schema may generate many goal patterns for searching assistant activities. For example, based on the RVL form instance (Figure 3.4), if one activity wants assistance for checking out the date when 'Howard' taught Oracle, the assistant activity slot might be specified as the following:

```
assistant-actvities:
      (for (date :in (formbase-query
                        :forms ((?rvl (instance-of RVL)
                                      (name Howard)
                                      (time-tab (subject Oracle))))
                       :subarea-path (time-tab)
                       :attribute date))
            :do (sub-goals (check-date date)))
```

It then will generate the following goal patterns:

```
(check-date  4/3/92)
(check-date  11/3/92)
(check-date  18/3/92)
(check-date  25/3/92)
```

These goal patterns do not assist each other, but they may compete for the same resources. In other words, conflicts may exist between them or their assistant activities. For example, in the warehouse example, in order to transfer p6 from site B to site A, we need a size 3 space; in order to find a size 3 space in the warehouse at site A, we can move p2 from s3 to s1, then move either p3 or p4 into s3. The goal pattern (move p3 A) and (move p4 A) are two son-requests of the request-node (find-space 3 A). Since there is only one size 2 space, only one of them, p3 or p4, can be moved. This is an example of the interactions between the formbase activities (request nodes). More detailed discussion about the interaction can be found in section 6.3.3.

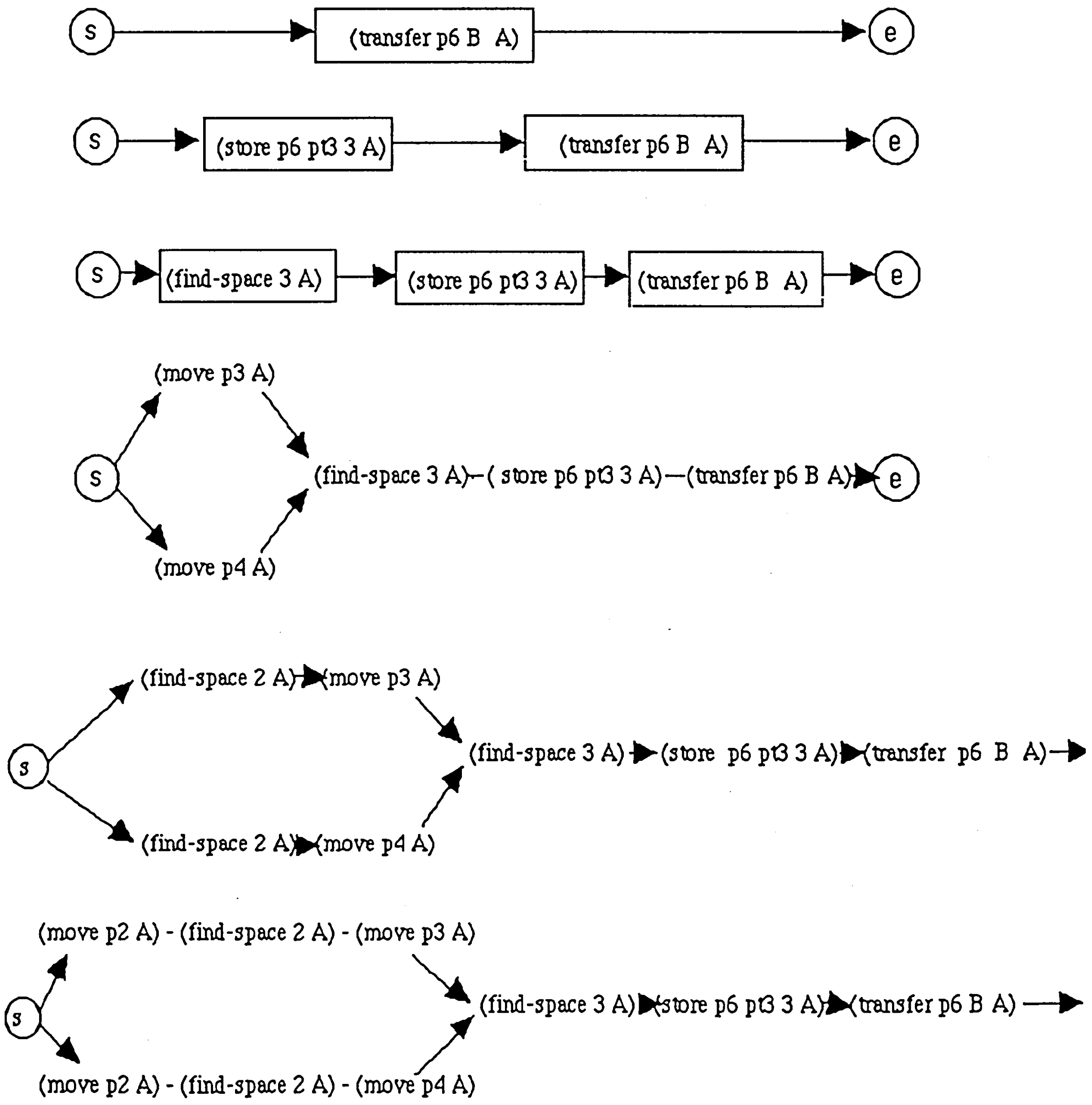


Figure 6.8

6.3. Problem Solving for the Activities upon Forms

In this section, the problem solving process for the activities upon forms is developed.

Whether a task, such as (transfer p6 B A), can be achieved by a IFS system depends on if the request-node of the goal pattern can be satisfied. The problem solving process uses a request network to explore the requests of the goal pattern and to resolve the conflicts.

The whole process has two phrases. The first phrase is to expand the request network until all the leaves of the network have been satisfied, the second phrase then tries to resolve the conflicts. Figure 6.8 shows an expanded request network for (transfer p6 B A) goal pattern. In order to resolve the conflicts of the expanded network, the effects of the activities must be evaluated.

6.3.1 Request Network

A **request network** is a network of the request-nodes. For example, Figure 6.8 is a request network for the activity of transferring p6 from site A to site B. A request network also has a **start node** and a **end node** . Except start node and end node, every **request-node** in the request network is labelled by its goal pattern, such as (transfer p6 B A), (store p6 pt3 3 A) and so on. Except **start node**, every node has at least one son-request, and except **end node**, every node is a son-request of another request-node. Request network is directed, the request at the tail of an arrow is requested by the one that is at the head.

The one at the tail is the **son-request** of the one at the head, similarly the one at the head is the **parent-request** of the one at the tail. For example, in Figure 6.8, (store p6 pt3 3 A) is the son-request of (transfer p6 B A), and both (move p3 A) and (move p4 A) are son-requests of the same request-node (find-space 3 A). Start node has no son-request, and end node has no parent-request.

Another important concept is called **Branch-request**. Except start node and end node, every request-node in the request network has a Branch-request. A **Branch-request** is the request-node which is at the head of a branch (direction defined by the arrows in the branch) in the request network, and it is the goal of the branch. For example, the Branch-request of the (move p2 A) node in the upper position in Figure 6.8 is (move p3 A). The Branch-request of (find-space 3 A) is end node. Obviously, if one request-node can not be satisfied, its Branch-request fails.

6.3.1.1 Node Definitions of Request Network

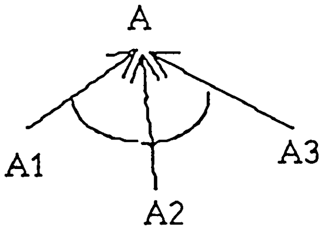
Three types of nodes have been defined for representation of a request network: they are start node, end node, and request node. The request-node contains following information:

node-id	: a system generated identifier;
goal-pattern	: the goal-pattern label of this node;
holding-cond	: constraints which represent the conditions which are not controlled by the knowledge of this node;
non-assistant-cond:	constraints which represent the conditions whose achieving assistant activity is specified in the assistant-activities slot of this node;

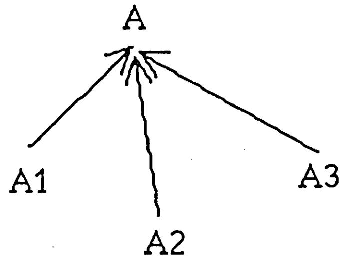
status	: Phantom (need not to extended), WEC, WE, WB, Expanding and AE;
current-act	: the activity-identifier of the activity currently used to expand the node;
activity-list	: the list of activities that can be used, but have not been used to expand the goal pattern;
operations	: if all the conditions are satisfied, the operation here will be performed upon the forms instances in the form variable slot;
next	: Request-nodes that follow this node;
prev	: Request-node that is precedent to this node;

6.3.1.2 Types of Request Node

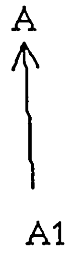
A request-node (a formbase activity) may require that all its son-requests be successful, or require that some of its son-requests be successful. If a request-node requires that all its sons be successful, the request-node is referred as **J-request-node**, otherwise, the request node is referred as **D-request-node**. If a request-node only has one son, it is also called a **request-link**. A J-request-node, a D-request-node or a request-link can be represented by diagrams as it is shown in Figure 6.9. In this Figure, A is a request-node (or an activity), A1, A2 and A3 are its son-requests.



J-request-node



D-request-node



request-link

Figure 6.9

The IFS system defines that a J-request-node is **satisfied** if all of its son-requests can be satisfied, and defines a D-request-node is **satisfied** if at least one of its son can be satisfied, and a request-link is **satisfied** only if its son can be satisfied. Obviously, an activity (a request-node) can be achieved if its son-requests can be satisfied.

6.3.2 Expansion of Request Network

Request network expansion is the first phrase of the problem solving upon forms. The algorithm of expansion is a recursive process. The aim of expansion is to make sure that every request-node follows the Start Node is in phantom status, that is, all the goal patterns which follow Start Node can be satisfied by the formbase directly. In the following, the warehouse example is used to present the expansion process.

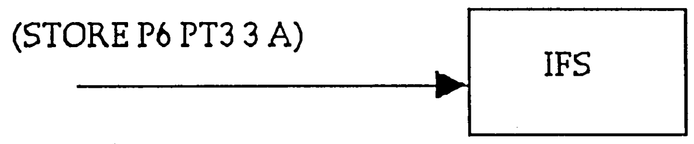


Figure 6.10

Suppose the initial goal pattern required is (store p6 pt3 3 A), see Figure 6.10. After the goal is input, the IFS system uses this goal pattern to search assistant activities which can assist the goal fulfilment. One of these activities will be chosen as 'current-act' to expand and to explore the details of the problem solving, the rest of them will be stored in 'activity-list' for backtracking. In this example, the activity representation that matches the request is <store-product> (see section 4 of Chapter 5), the system then get an initial request network shown in Figure 6.11. This request network means that the request (store p6 pt3 3 A) can be fulfilled by performing activity <store-product>. In this example, <store-product> is the only activity that can fulfil the goal pattern, therefore the 'activity-list' of this node is empty.

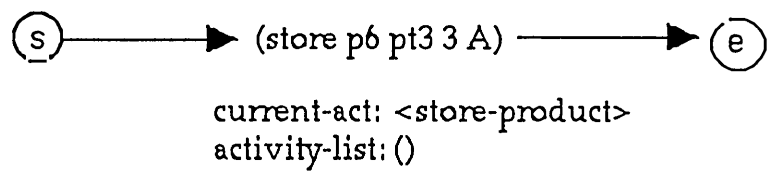


Figure 6.11

In Figure 6.11, the goal pattern (store p6 pt3 3 A) is used to label the request-node, but the actual content of this node contains the following information:

```
node-id            : Fam-1

goal-pattern       : (store  $product-id  $product-type $space-required $site)

form-variable      : ( ($product-id p6) ($product-type pt3)
                       ($space-required 3) ($site A) )

holding-cond       : ()

non-assistant-cond     : (some-bigger
                      :forms  ((?ts (instance-of TS)
                                    (site $site)
                                    (space-tab (status Free))))
                      :subarea-path (space-tab)
                      :attribute space-size
                      :value $space-required)

status             : Unexpanded;

current-act        : <store-product>
```

```
activity-list       : ()

operations          :
        (let ((sp-id
                (one-of (formbase-query
                          :forms ((?ts (instance-of TS)
                                       (site $site)
                                       (space-tab
                                       (status Free)
                                       (space-size >=$space-required))))
                          :subarea-path (space-tab)
                          :attribute space-id))))
          (set-value :form ((?ts (instance-of TS)
                                  (site $site)
                                  (space-tab (space-id sp-id))))
                      :subarea-path (space-tab)
                      :attribute status
                      :value Free)
         (add-subform
                :forms ((?tsp (instance-of  TSP)
                              (site  $site)))
                :subarea-path (ware-house-tab)
                :value ($product-id $product-type pt-id (date))))

next                : End-Node

prev               : Start-Node
```

The work of creating the node is done by the expansion process, in the following only the goal patterns are used to label the nodes, the content of the nodes will be omitted.

In Figure 6.11, the <store-product> activity has a non-assistant-condition to search if there is a space which is equal or bigger than the requirement of the product. If suppose that the current situation in site A warehouse is as shown in Figure 6.5, then there is no such spaces available, therefore more assistant activities will be searched. The <store-product> activity has a son-request which is a request-link for searching assistant activities. In this case, the goal pattern for the request-link is (find-space 3 A). The system then uses this goal pattern to search activities for expanding the network. There is again only one activity that matches the goal pattern, that is <find-space>. Therefore, the request network can be expanded as shown in Figure 6.12.

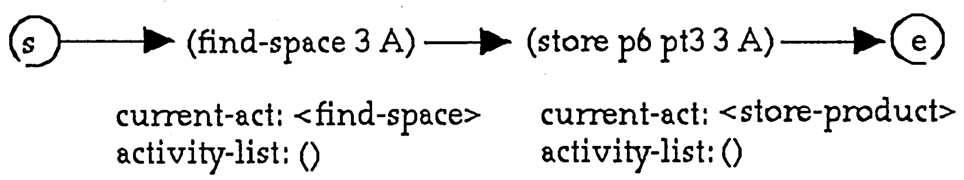


Figure 6.12

Notice that <find-space> activity has holding conditions which at this moment can be satisfied by the situation. If the holding conditions can not be satisfied, then <find-space> activity can not be applied. Since there are no other activities in the 'activity-list' of the (find-space 3 A) request-node, (find-space 3 A) request can not be satisfied. Therefore IFS system will backtrack to find if there is an alternative activity representation to expand the Figure 6.10 request network. There is no other activities in the 'activity-list' for (store p6 pt3 3 A) node, the (store p6 pt3 3 A) also can not be satisfied. The goal request of (store p6 pt3 3 A) is End Node, therefore if the holding condition of <find-space> activity could not be satisfied, the goal pattern (store p6 pt3 3 A) would fail.

Since the holding conditions of <find-space> can be satisfied, the system continues to expand the request network. The goal patterns for searching assistant activities in this case are (move p3 A) and (move p4 A). The <find-space > activity does not require all of them be successful, therefore (find-space 3 A) is a D-request-node of the two son-requests. See Figure 6.13.

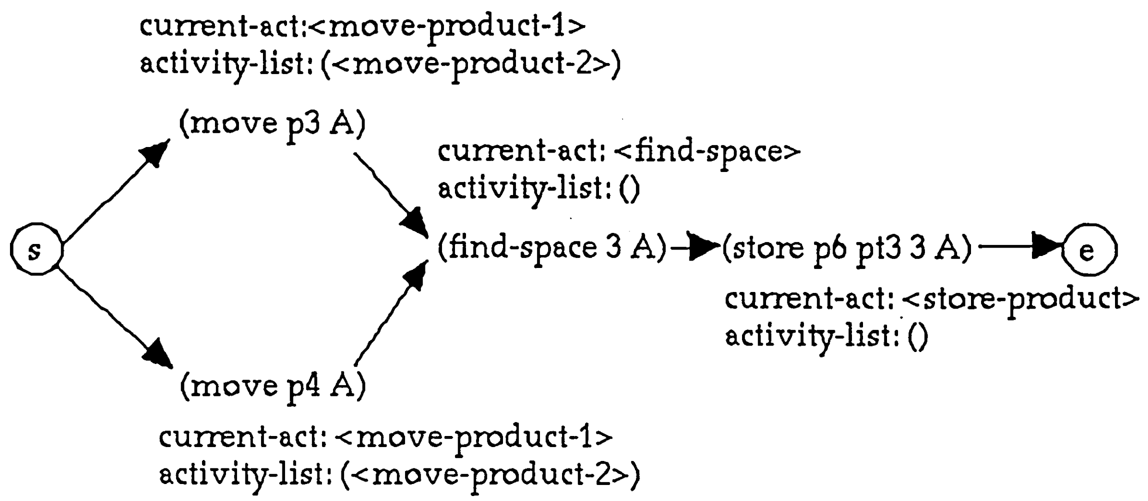

current-act:<move-product-1>
activity-list: (<move-product-2>)
(move p3 A)
current-act: <find-space>
activity-list: ()
s
(find-space 3 A)
(store p6 pt3 3 A)
e
current-act: <store-product>
activity-list: ()
(move p4 A)
current-act: <move-product-1>
activity-list: (<move-product-2>)

Figure 6.13

Following the same principle, this request network can be further expanded until all the request-nodes next to the Start Node can be set to phantom status, see Figure 6.14.

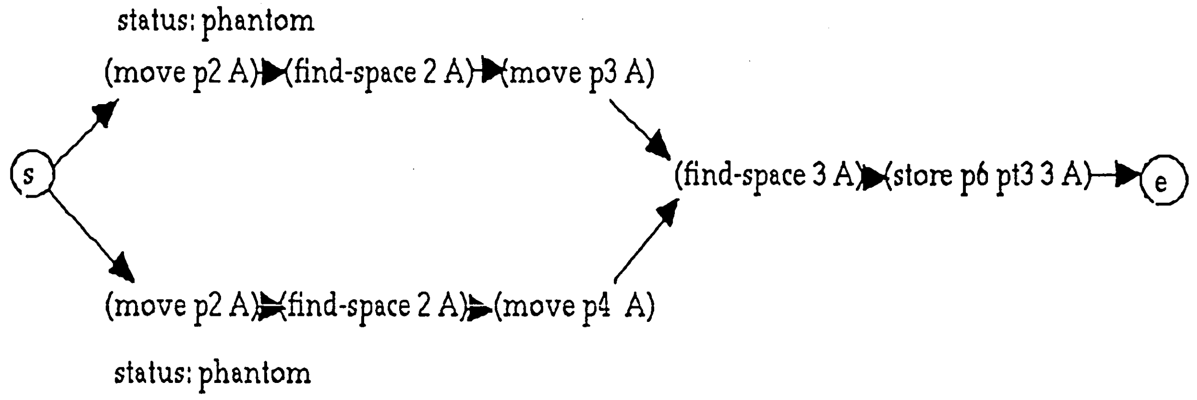


Figure 6.14

The expansion algorithm is mainly composed by two processes. One is expansion, the other is backtrack. These two processes recursively call each other. The backtrack of the expansion happens when there is no activity representation available for expanding a request-node. Generally the backtrack process checks if there are any alternative activity representations in the 'activity list' in the parent-request node for its goal pattern. If there are, it re-expands the request network with another activity representation. If there isn't any, it continues to backtrack until a Branch-request is encountered. It then checks if the parent-request node of the Branch-request is a D-request-node, if it is a D-request-node, this branch will simply be discarded from the network, otherwise the parent-request node of the Branch-request can not be satisfied.

6.3.3 Conflicts Resolution

The expansion process ends when every request-node that follows the Start Node is in phantom status, but this is not the end of the problem solving, serious conflicts may still exist. For example the expansion for (store p6 pt3 3 A) request will end with a request network that is shown in Figure 6.14, but conflicts are still existing in the network, (move p3 A) and (move p4 A) nodes are conflicting to each other since only one pt2 type product can be moved into space s3. In this section, the process of resolving the conflicts is introduced.

6.3.3.1 When and Why to Conflict

Before getting into the details of the algorithm, in this section the condition of the conflicts is studied. A form is consciously designed for the management of an organization. It functions as a window or a filter for office workers to observe and to control the activities inside the organization. That is, what data to perceive and how to perceive it are usually defined when the type of an form is designed. Every time of perception or manipulation of a special portion of data may require that some data had been perceived or some manipulations had been carried out.

In another words, every activity may require assistant activities. According to the design, an assistant activity will not override the effects of its parent activities since they take charge of different portions of data inside the forms. However, the activities which are not to assist each other may try to fill an attribute with different values. If an attribute is single valued, this would cause confusion for data perception and disorder for the data manipulations.

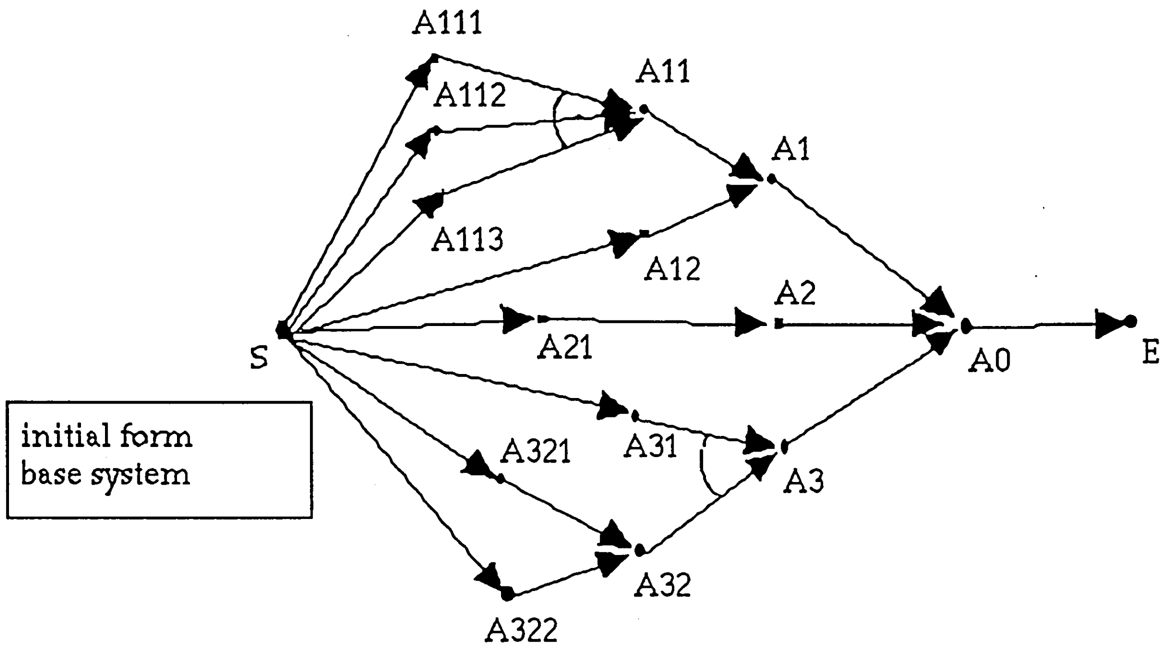


Figure 6.15.

This type of conflict can easily be sensed if the system keeps a record of which activity adds a value to which attribute. From Chapter 4, we know that every attribute belongs to an instance of an abstraction. Therefore, IFS uses the following structure to record the effects of every activity, it is called **Form Effect Token (FET).**

```
<FET>
{
 instance-id:
 attribute:
 attibute-type:
  value:
 activity:
 node-id:
}
```

The 'attribute-type' slot specifies whether the attribute is multiple valued or single valued. A conflict happens if two activities try to give different values to one attribute which is single valued.

6.3.3.2 Where to Conflict

After expansion, the request network is a directed graph. Every arrow in the network directs from the Start Node to the End Node, indicating the actual order of the activities. A diagram of a request network is shown as Figure 6.15. Since every activity in the network follows the organizational activity assumption, every activity at the tail of the arrow supports the activity at the head of the arrow, therefore an activity at the tail of the flow will not conflict with the activities at the head of the flow. For example, A21 will not conflict with A2 and A0, and A322 will not conflict with A32, A3 and A0. However, sons and the posteriors of one request-node may conflict with each other. For example, A113 may conflict with A112, A111, and with A12, A31, and etc.

If a conflict is existing between two sons of a J-request-node, this conflict can not be resolved, the J-request-node can not be satisfied. For example, if A111 is conflicting with A112 or A113, since they belong to a J-request-node, both of them has to be true at the same time, therefore the conflict can not resolved. Usually to resolve a conflict between two activities, one of the activities has to be abandoned, in other words, only one of them can be satisfied. For example if A321 is conflicting with A322, then one of them has to be abandoned. For every request-node in the request network, as long as there is a path which is composed of the request-nodes of its assistant activities from the Start Node to it, and the conflicts are resolved, the goal pattern of this node can be achieved. If after resolution, there is no such a path, this node can not be satisfied.

Therefore, if all the conflicts of an expanded request network can be resolved, the left network can be executed to achieve the initial goal.

6.3.3.3 Conflicts Resolution and Goal Re-Commitment

In a request network, though every activity will not conflict with its assistant activities, there is still a large number of activities which may conflict with each other. Once a conflict is identified through FET (Form Effects Token), it needs to be resolved. Some conflicts can not be resolved, therefore they will cause re-commitment of the goal pattern. If a conflict can be resolved, one of the node of the conflict will be abandoned.

6.3.3.3.1 Relations for Conflict Resolution

To develop the conflict resolving process, the concept of node dependency is developed first. A request-node R1 is **dependent on** another request-node R2 if and only if the failure of R2 will cause the the failure of R1. In this case, we also say R1 is a **dependant** of R2. In a request network, inside one branch the request-node at the head always depends on the request-node at the tail. For example, in Figure 6.16(a) node B is dependent on node A, and node C is dependent on both node A and node B. Obviously in one branch of a request network, the dependent relation is transitive. B is dependent on A, C is dependent on B, C is also dependent on A. Therefore, the branch-request node is always dependent on the other nodes of the same branch.

$$\begin{tikzcd}
A \arrow[r] & B \arrow[r] & C
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
A \arrow[r] & B \arrow[r] & C \arrow[dr] & & & \\
& & & E \arrow[r] & G \arrow[dr] & \\
& & F \arrow[ur] & & & H \\
& & & & I \arrow[ur] &
\end{tikzcd}$$

(b)

Figure 6.16

If the parent-request node of a branch-request node is a J-request-node, all the nodes that are in the same branch of the J-request-node will be dependent on the nodes which are in the same branch as the branch-request node. For example in Figure 6.16(b), node E is a J-request-node, therefore node E and node G are dependent on node A, B, C and node F. However, node H is a D-request-node, therefore node H is no longer dependent on either node G or node I.

Another concept of request-node which is defined based on the dependent relation is called **farthest dependent node**. As we can see from

Figure 6.16(b), node G is dependent on node A, but node H is no longer dependent on it, therefore node G is the farthest dependant node of node A in the network. So a request-node R1 is the **farthest dependent node** of another request-node R2 if and only if the parent-request node of R1 is no longer dependent on A. A farthest dependent node of a request-node could be the node itself, for example, the farthest dependent node of node I in Figure 6.16(b) is node I itself.

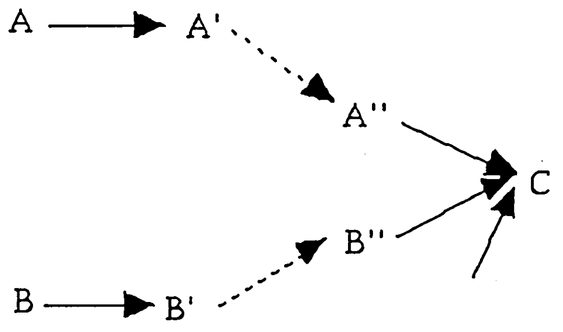


Figure 6.17

If a conflict is identified between node A and node B as it is shown in Figure 6.17, the conflict resolving process first searches out the nearest node which requests both of them, which is called the **nearest common receiver** of A and B. In Figure 6.16(b), the nearest common receiver of node A and node F is node E, while the nearest common receiver of node A and node I is node H.

6.3.3.3.2 Condition for Conflict Resolution

In Figure 6.17, suppose that the nearest common receiver of node A and node B is node C, in order to resolve the conflict between A and B, the conflict resolving process checks whether node C is dependent on node

A and node B. If node C is dependent on both of them, the conflict between A and B can not be resolved, therefore node C can not be satisfied. If node C is dependent on one of them, then the node which C is not dependent on will be abandoned to resolve the conflict. If node C does not depend on either of them, then an interactive process will be used to ask user to choose which one will be abandoned.

6.3.3.3.3 Process of Abandon

If a request-node in a request network is asked to be abandoned, the processing is simple. The conflict resolving process searches out the farthest dependent node of the abandoning node, and cut all the request-nodes that are requested by it and itself away from the request network.

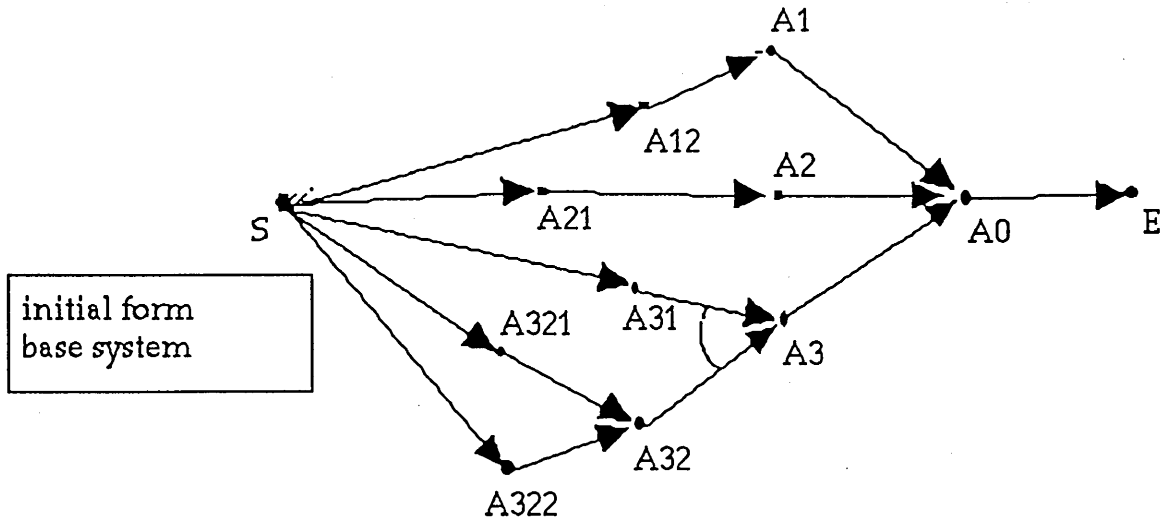


Figure 6.18

For example, in Figure 6.15 if the system abandons node A111, it first searches out the farthest dependent node of A111, which is A11 in the example. It then cuts A11 and all the nodes that are requested by A11 away. The left request-network is shown as Figure 6.18.

6.3.3.3.4 Re-Commitment

If a conflict can not be resolved, the nearest common receiver of the two conflicting nodes can not be satisfied by this expansion, the goal pattern of the request-node will be re-commited to the system. The re-commitment is another expansion process which starts from the failed goal pattern. For example, if in Figure 6.15 the node A31 and node A32 are conflicting to each other, since their nearest common receiver A3 is dependent on both of them, the conflict can not be resolved by this expansion. Therefore re-commitment of the A3 is required, as it is shown in Figure 6. 19. Here A3 is not in Phantom status, therefore needs to be expanded.

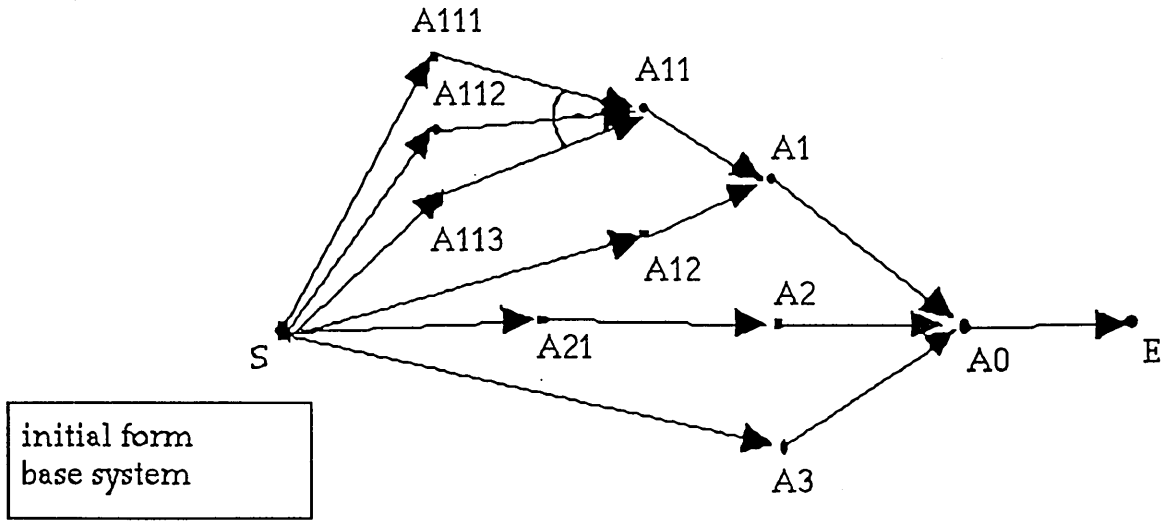


Figure 6.19

Obviously the expansion may again involve backtracking. If the re-commitment succeeds, the request network will be again handed in for checking conflicts.

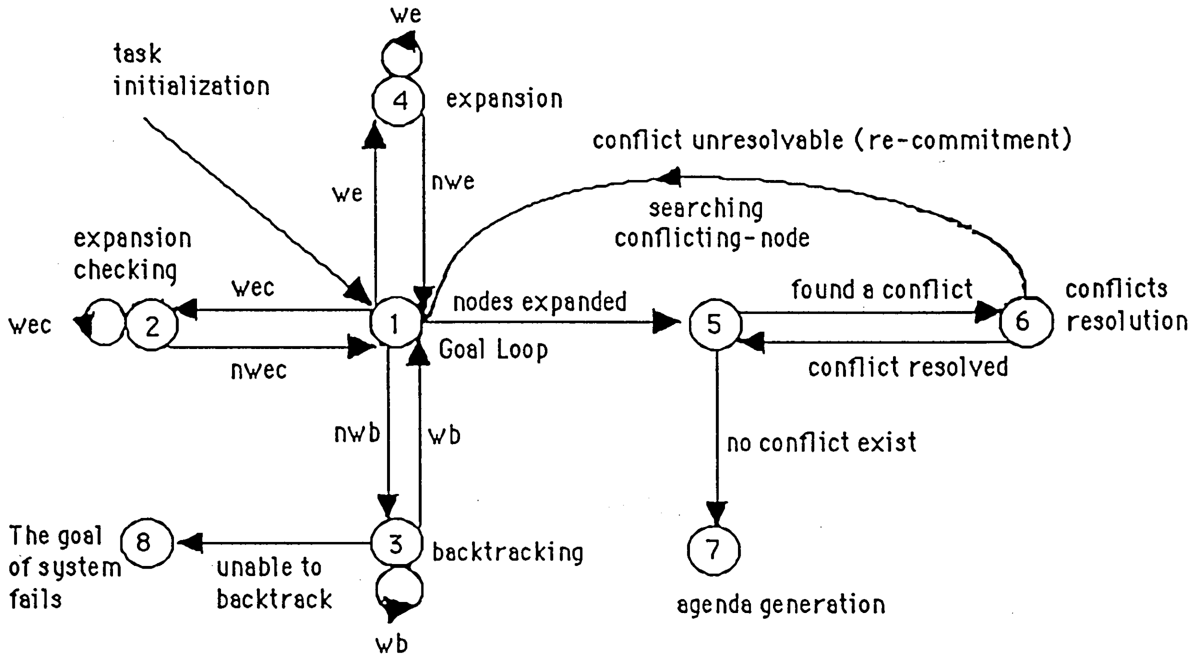
Once the re-commitment fails, the problem solving process for the initial goal fails. This will be a failure end of the problem solving.

6.4 The Control Structure of the Formbase Activity Problem Solver

This section presents the control structure of the problem solving for the activities upon the forms. The problem solver has been fully implemented by using Common Lisp in Sun-3 workstation.

Different from the developed planner such as NONLIN [Tate, 1977] and TWEAK [Chapman, 1987], since the interactions of the subgoals of the formbase activities can not be evaluated during the network expansion, due to the inability of the pattern matching of the form reference pattern language (see section 3.8), the problem solving of the formbase activities can not be controlled by a searching process such as Dependent Backtrack Searching used by TWEAK. However, since the formbase activities obey the Organizational Activity Assumption, the needed step during the problem solving process can be determined by the node status of the nodes in the request network, therefore an automata can be designed to control the problem solving process. The state graph of the automata is shown as Figure 6.20.

The automata contains 8 states, the state information is stored in the request-network which is defined as a global variable and can be accessed by the procedures that are attached to the states. The status field of every node in the request network may take 5 different values: they are "WEC" which stands for Waiting for Expansion Checking, "WE" which stands for Waiting for Expansion, "WB" which stands for Waiting for Backtracking , "AE" which stands for After Expansion and "Phantom" which means the node needs not expansion.



The State Graph of the Control Automata

Figure 6.20

Basically when a list of activities is searched out against a goal pattern, a node is generated attaching on the list of activities. The node is then in the status of WEC (waiting for expansion checking), the expansion-check process will check if there is an activity whose holding-conditions are matched by the formbase, if there is one, then the information of this activity will be installed on the node and the status of the node is set as WE (waiting for backtracking). If however there is no activity whose holding-conditions are true (matched by the formbase), the status of the node will be set as WB (waiting for backtracking), then the backtracking process will backtrack the expansion process from this node.

When a task has been initialized, the request network contains nodes which are in status of "WEC", such as the network shown in Figure 6.11. The procedure of State 1 in Figure 6.20 then takes control of the problem solving process based on the request network. The pseudo code of the procedure is shown as the following:

```
Procedure goal-loop
 begin
    while (t)
      begin
           if (all the leaves of req-net are in Phantom status)
               then exit while
              else begin
                     case (status of the nodes of the request network)
                        there is WEC node: expansion-checking;
                        there is WB node: backtracking;
                        there is WE node: expansion;
                     end {case}
              end {else}
        end {while}
    search conflicting nodes;
 end; {procedure}
```

Therefore if there are nodes which are in WEC status, the system will carry on expansion checking; if there are nodes in WB status, the system will carry on backtracking, and if there are node in WE status, the system will carry on expansion process. Every process is itself an iteration, so unless there is no node in the network that is in the status of the process corresponding to, the process will not return control to Goal-Loop process.

For example, the expansion-check process uses the following pseudo algorithm.

```
Procedure expansion-check
 begin
    while (t)
      begin
          if  (there is no WEC node)
               then exit while
               else expansion-checking;
      end {while}
 end; {procedure}
```

6.5. Summary

In this chapter, the specialties of the problem solving upon forms are discussed. A problem solving process for the activities upon the forms is developed. The problem solving process involves expansion, backtracking, conflict resolution and re-commitment. It provides a mechanism for office form user to search out possible assistant activities and resolve the conflicts between these assistant activities.

Chapter 7.

Research Conclusions

This chapter reviews the research aim, and summarises the contributions and the unsatisfactory aspects of the system. A suggestion regarding further research is also presented.

7.1. Review of Research Aim

This research is based on earlier developments in Office Information Systems such as OBE (Office-By-Example) [Zloof, M., 1982], SCOOP [Zisman, 1978], ICN [Ellis, C., 1979], OMEGA [Barber, 1983], POISE [Croft & Lefkowitz, 1984], POLYMER [Croft & Lefkowitz, 1988], OFM [Tsichritzis, D. C., 1982], SOS [Bracchi, 1984] and OPAS [Lum, V. Y., 1982]. The aim is to improve the flexibility and stability of an Information System inside which data as well as the procedures that manipulate the data can not be pre-defined. The key issue of the aim is to identify a mechanism so that dynamic task requirements can be fulfilled, based on partially well defined procedural knowledge and the situation.

The methodology that this research has taken is to apply the structure of an AI planner to the system modeling, that is, the situation is modelled by a database while the activities are modelled by activity schemas. In an open system, knowledge of the activities can only be partially defined, the approach of using activity schemas (task formalisms) to model the activities therefore fits the semantic requirements. However, the information processing activities are different from the activities of a robot, the application of AI Planning systems to such a system is not at all straightforward, much effort has been directed towards research in this area.

By taking this methodology the research has successfully identified a mechanism so that the aim can be achieved. That is, when constructing a system, the mechanical analysis of the system is no longer needed, instead only partially well defined knowledge needs to be identified. Therefore when changes take place in the environments, alterations are only needed for the related portion of knowledge.

However, the representational methods of activities has not yet identified a "neat" language, the activity representation is still trivial, programming language related and very un-friendly to user. These features are unfortunately results of the un-normalised nature of forms, therefore in order to improve the usability of the system, further research is needed to explore a graphical interface for the system.

7.2 Contributions of the Research

In order to construct the Intelligent Form System, the following problems must be considered: 1) how to represent user's task requirements; 2) how to represent the activities upon forms; 3) how to define and represent the primitive activities upon forms; and 4) how to generate the activity sequences for achieving the task requirement of a user.

Three major efforts have been made for developing the Intelligent Form Systems. They are: a) the development of the Form Reference Pattern and the Formbase System so that data of forms and primitive operations can be represented; b) the identification of the formbase activity schema to represent the activities which are not primitives; c) the identification of the Organizational Activity Assumption and the conflict resolving mechanism, on which the problem solving mechanism for a task can be developed.

A prototype of the system has been constructed on Sun-3 Workstation, which is able to demonstrate the process of task representation, decomposition and fulfilment in a practical situation.

7.2.1 Formbase System

The Formbase System developed enables us to represent as well as to manipulate the data in a more complex data structure ---- form. A form is a data entity that is normalised. The Form Reference Pattern Language that has developed in the formbase supports a complete set of operations which include data definition as well as data manipulations. Compared to Codd's relational calculus or SQL, and even the more recent DAPLEX and OSQL, a system with such specifications has extended our ability to directly manipulate the data inside forms. In contrast to the Object-Oriented Database Systems, the Formbase System does not aim to be a generic data modellingsystem, it does not support inheritance relations as well as procedure attachment mechanisms, therefore the storage structure and implementation can be largely simplified.

7.2.2 Formbase Activity Schema

The formbase activity schema can represent the activities that process two types of representations, namely predicate representation and form reference pattern representation. Thus the information processing activities upon forms can be directly represented by the activity schemas. The significance of this is that the activity representation can therefore be situated in the forms. For open information system modeling, this ensures the stability of conceptual modeling, since when activities are represented purely in terms of predicates, the representations are not situated in the information entities of the system. Instead, because predicates can be freely chosen, the activity descriptions are superficial and random, the system modellingtherefore runs a big risk of uncertainty.

It is the definition of formbase activity schema that opens the possibility of applying the AI problem solving technologies to problem solving for activities upon forms.

7.2.3 Organizational Activity Assumption

The Organizational Activity Assumption can be described in one sentence ------ **in a procedure network an activity will not override the effects of the activity which searches it out**. The activities will not override the effects of those activities that search them out, the extension of the network (which is actually a tree) therefore need only be considered at the leaf-level. With the identification of the condition of conflicts resolution, the problem solving process for the formbase activities can be constructed.

It is the Organizational Activity Assumption that distinguishes problem solving for the information activities upon forms from the unstructured activities which are performed automatically by the human. Since organizational activities exist, this assumption reveals a cognitive process which is on-going but which has not been formally recognized. Based on the Organizational Activity Assumption, problem solving for the formbase activities is a process of conflict resolving for a network of requests.

7.3 Shortcomings and Further Research

The shortcomings and proposals for further research for the IFS are addressed in this section.

7.3.1 Shortcomings of the System

The prototype of the system has successfully demonstrated the ability of the system to generate an activity sequence for the required task based on knowledge of the partially well defined activities. However, the form reference pattern language and the activity representation methods still have shortcomings that need to be improved.

7.3.1.1 Limitations of the Form Reference Pattern

The FRP (Form Reference Pattern) has shortcomings which can be introduced by comparing it to the pattern of Prolog. For example, in Prolog the following pattern represents variable **Y** is **Jim**'s father and **X** is **Y**'s father and **Jim**'s grandfather.

```
?- parent(Y, Jim), parent(X, Y)
```

In this pattern, not only is the variable **Y** in pattern **parent(Y, Jim)** the same as the Y in **parent(X, Y)**, but there is also a close link between **Y** and **Jim**, **Y** and **X** so that they have to be satisfied together. This convention is not however conserved in the Form Reference Pattern. For example, look at the following reference pattern.

```
((?ts (instance-of TS)
      (location A)
     (space-tab (status Occupied))
     (space-tab (space-size 3))
     (space-tab (space-id ?sp-id)))
 (?tsp (instance-of TSP)
        (site A)
        (warehouse-tab (space-id ?sp-id))))
```

This pattern searches the TS forms (Figure 6.6) which are in location 'A', and have occupied spaces and also have size 3 spaces. It also requires that the TS forms have corresponding TSP forms (Figure 6.7) which have at least one space 'id' the same as the space in the TS forms. But in the TS forms that are searched out, whether all the size 3 spaces are Occupied or not is not clear, since the constraints are not accumulatively required. For the same reason the variable ?sp-id only represents the space identifiers that satisfy the relational constraint of TS and TSP. It does not represent the spaces whose size is 3 and are occupied. For example, based on Figure 6.6 and Figure 6.7, the ?sp-id variable in above pattern will refers to S2, S3, S4 and S5, only S4 and S5 have size 3.

This feature of the form reference pattern would cause problems for queries in certain situations when variables are needed in sub-condition specifications, and when constraints for the attribute exist in other tables. When the constraints of the attribute are existing in the same form, the query is still able to give out the right results. For example if we want to query the space which is occupied and is of size 3, and also has connection with the TSP form, we can have the following query:

```
(formbase-query
      :forms
         '((?ts (instance-of TS)
                (location A)
               (space-tab (status Occupied))
               (space-tab (space-size 3))
               (space-tab (space-id ?sp-id)))
          (?tsp (instance-of TSP)
                (site A)
                (warehouse-tab (space-id ?sp-id))))
     :type 'TS
     :subarea-path '(space-tab)
     :sub-cond '((status Occupied) (space-size 3) (space-id ?sp-id))
     :attribute 'space-id)
```

In this query, the ?sp-id variable is used, but since ?sp-id does not accumulate the results of (space-size 3) and (status Occupied), they are specified again in the sub-condition specification. This query is able to give what we want. However, if we want the product identifier in the space which is of size 3, the direct result cannot be given out by the formbase-query function. For example, if we use the following query, since ?sp-id variable in this query actually has values S2, S3, S4, S5, we are not able to get what we want.

```
(formbase-query
      :forms
         '((?ts (instance-of TS)
                (location A)
               (space-tab (status Occupied))
               (space-tab (space-size 3))
               (space-tab (space-id ?sp-id)))
          (?tsp (instance-of TSP)
                (site A)
                (warehouse-tab (space-id ?sp-id))))
      :type 'TSP
      :subarea-path '(warehouse-tab)
      :sub-cond '((space-id ?sp-id))
      :attribute 'product-id)
```

When we use this system, since the prototype is implemented in Lisp, we can finish this query by the help of Lisp, shown as the following:

```
(let ((spid-list (formbase-query
        :forms
           '((?ts (instance-of TS)
                  (location A)
                  (space-tab (status Occupied))
                  (space-tab (space-size 3))
                  (space-tab (space-id ?sp-id)))
             (?tsp (instance-of TSP)
                   (site A)
                   (warehouse-tab (space-id ?sp-id))))
      :type 'TS
      :subarea-path '(space-tab)
      :sub-cond '((status Occupied) (space-size 3) (space-id ?sp-id))
      :attribute 'space-id)
   (for  (sp-id :in spid-list) :do
      (formbase-query
      :forms
         '((?ts (instance-of TS)
                (location A)
                (space-tab (status Occupied))
                (space-tab (space-size 3))
                (space-tab (space-id ?sp-id)))
           (?tsp (instance-of TSP)
                 (site A)
                 (warehouse-tab (space-id ?sp-id))))
      :type 'TSP
      :subarea-path '(warehouse-tab)
      :sub-cond '((space-id sp-id))
      :attribute 'product-id)))
```

To improve this complexity, the pattern language has to be augmented with a more complex structure, or with more calculus. For example, '@i' may be introduced to mark the constraints that must be satisfied accumulatively. Therefore, we may have:

```
((?ts (instance-of TS)
      (location A)
      @1(space-tab (status Occupied))
      @1(space-tab (space-size 3))
      @1(space-tab (space-id ?sp-id)))
 (?tsp (instance-of TSP)
       (site A)
       (warehouse-tab (space-id ?sp-id))))
```

In this pattern, the constraints that are marked by '@1' mean that they should be satisfied accumulatively.

7.3.1.2 The Complex Roles of Variables in Activity Representation

The above shortcomings of the Form Reference Pattern causes the complexity in activity representation when using the five-key system to query or manipulate the formbase. The roles of the variables in activity representation is also very complex, therefore at the moment the activity representation is lengthy and Lisp programming language related. For example in the following, to represent the activity of moving a product in warehouse management, there are many places where a quote (') or a backquote (`) and a comma (,) have to be used. This is because in representing an activity, the roles which a variable will play in the expression are very complex.

```
<moving-product>
{
   goal-pattern: '(move $product-id $site)
   supervised-cond: ;; check if there is free space which is bigger than
                    ;; the space that the product requires.
         '(some-bigger
              :form1 '((?ts (instance-of TS)
                            (site $site)
                            (space-tab (status Free)))
```

```
            :subarea-path1 (space-tab)
            :attribute1 space-size
            :form2 '((?tsp (instance-of TSP)
                           (site $site)
                           (ware-house-tab (product-id $product-id)
                                           (product-type ?pro-type)))
                     (?tp (instance-of TP)
                          (product-tab (P-type ?pro-type))))
            :type2  'TP
            :subarea-path2 '(product-tab)
            :attribute2 'space-required )
:assistant-activities
    `(let ((sp-size (formbase-query
                      :forms '((?tp (instance-of TP)
                                    (product-tab (P-type ?pro-type)))
                               (?tsp (instance-of TSP)
                                     (ware-house-tab
                                      (product-id $product-id)
                                      (product-type ?pro-type))))
                      :type 'TP
                      :subarea-path '(product-tab)
                      :attribute 'space-required)))
       (sub-goals 'find-space ,sp-size ,$site)))
:effects
    '(let* ((old-sp (formbase-query  ;; space the product stored
                      :forms '((?tsp (instance-of TSP)
                                     (site $site)
                                     (ware-house-tab
                                      (product-id $product-id))))
                      :subarea-path '(ware--house-tab)
                      :attribute 'space-id))
            (req-sp (formbase-query  ;; the space-size the product required
                      :form '((?tsp (instance-of TSP)
                                    (site $site)
                                    (ware-house-tab
                                      (product-id $product-id)
                                      (product-type ?pro-type)))
                              (?tp (instance-of TP)
                                   (product-tab (P-type ?pro-type))))
                      :type  'TP
                      :subarea-path '(product-tab)
                      :attribute 'space-required )
            (new-sp (one-of   ;;  space the product will move to
                      (formbase-query
                        :forms `((?ts (instance-of TS)
                                      (site $site)
                                      (space-tab (status Free)
                                                 (space-size ,rep-sp))))
```

```
                    :subarea-path '(space-tab)
                    :attribute 'space-id)))

                :

                :

}
```

7.3.2 Further Research

Despite the above described shortcomings, the conceptual model of the IFS has demonstrated a full cycle of user-centred information processing. The complexity of data that the activities can access and the nature of the activities are all more advanced than the systems that have been developed so far. The usability of the system then largely depends on whether there is a graphical interface so that the complexity of using the five-key system, and representing the activities upon the formbase can be reduced.

Appendix 1.

Examples of the Formbase Manipulations

This appendix lists a demonstration of the formbase manipulations. The system has been implemented by using Common Lisp in Sun-3 Workstation. The manipulations of this demonstration are made upon the RVL (Figure 3.1) and the ARVL (Figure 3.2) forms. In current system the function "list-form-instances" lists all the values of a set of form instances. For example,

```
(list-form-instances
            :forms '((?rvl (instance-of rvl)))
            :type 'rvl)
```

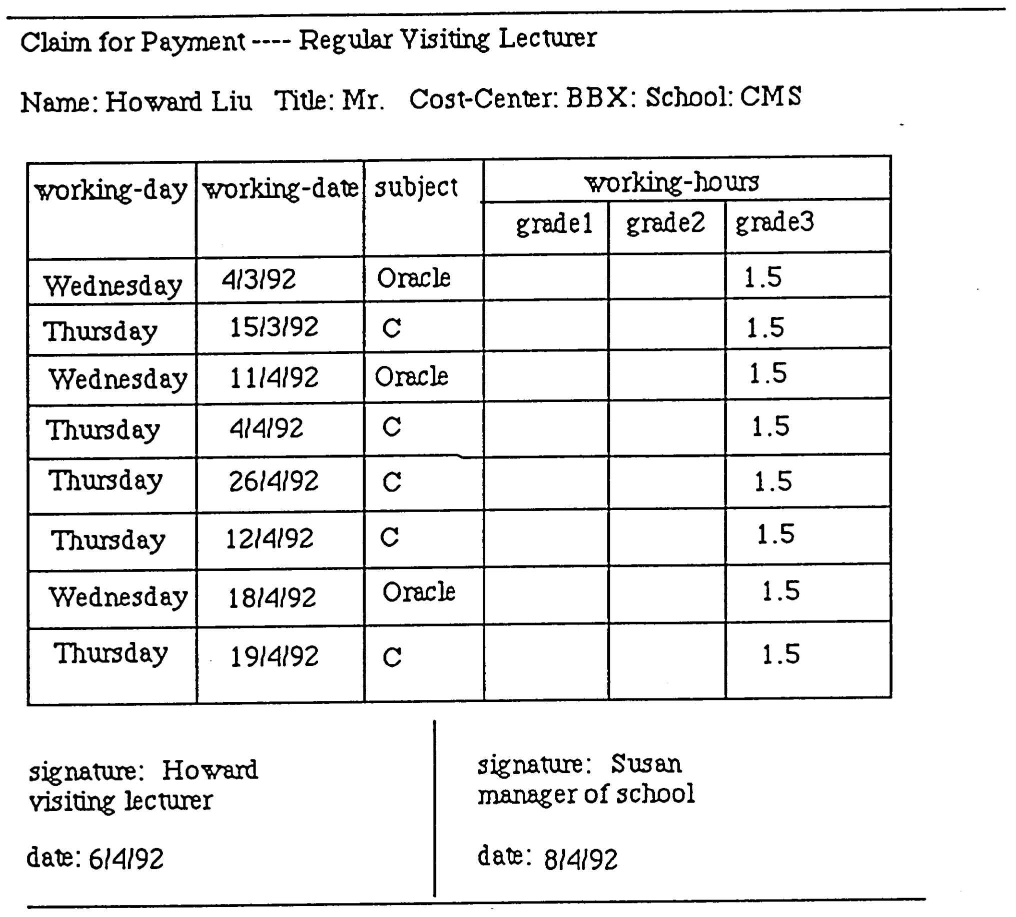
will list all the values of the RVL form instances in the current formbase. When "list-form-instances" function lists the values of a form instance, the subarea-path of the schema inside which the values are stored is listed first. For example, if there are three RVL form instances, shown as Figure A1.1, Figure A1.2 and Figure A1.3, stored in the formbase, the above query will give the following results.

```
> (list-form-instances
            :forms '((?rvl (instance-of rvl)))
            :type 'rvl)

 --------------- form instance: instan181
/RVL
NAME: HOWARD TITLE: MR. SCHOOL: CMS COST-CENTRE: BBX
/RVL/MANAGER-SIGNATURE
DATE: 8/4/92 MANAGER-SIGNATURE: SUSAN
/RVL/LECTURE-SIGNATURE
```

```
DATE: 6/4/92 LECTURER-SIGNATURE: HOWARD
/RVL/TIME-TAB
WORKING-DATE: 4/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 26/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 15/3/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 4/3/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 19/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 18/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 12/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 11/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
--------------- form instance: inst181
/RVL
NAME: CHUCK TITLE: MR. SCHOOL: CMS COST-CENTRE: BBX
/RVL/LECTURE-SIGNATURE
DATE: 6/4/92 LECTURER-SIGNATURE: HOWARD
/RVL/MANAGER-SIGNATURE
DATE: 8/4/92 MANAGER-SIGNATURE: SUSAN
/RVL/TIME-TAB
WORKING-DATE: 19/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
```

```
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 15/3/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 26/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 12/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 18/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 4/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 11/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 4/3/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
--------------- form instance: instan229
/RVL
NAME: TONY TITLE: MR. SCHOOL: CMS COST-CENTRE: CCX
/RVL/MANAGER-SIGNATURE
DATE: NIL MANAGER-SIGNATURE: NIL
/RVL/LECTURE-SIGNATURE
DATE: 10/7/92 LECTURER-SIGNATURE: TONY
/RVL/TIME-TAB
WORKING-DATE: 25/5/92 WORKING-DAY: MONDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 3 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
```

```
WORKING-DATE: 15/5/92 WORKING-DAY: THURSDAY SUBJECT:
BATABASE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 2 GRADE2: NIL GRADE1: NIL
-------------------------------------
NIL
>
```



Claim for Payment ---- Regular Visiting Lecturer

Name: Howard Liu Title: Mr. Cost-Center: BBX: School: CMS

working-day	working-date	subject	working-hours		
			grade1	grade2	grade3
Wednesday	4/3/92	Oracle			1.5
Thursday	15/3/92	C			1.5
Wednesday	11/4/92	Oracle			1.5
Thursday	4/4/92	C			1.5
Thursday	26/4/92	C			1.5
Thursday	12/4/92	C			1.5
Wednesday	18/4/92	Oracle			1.5
Thursday	19/4/92	C			1.5

signature: Howard
visiting lecturer

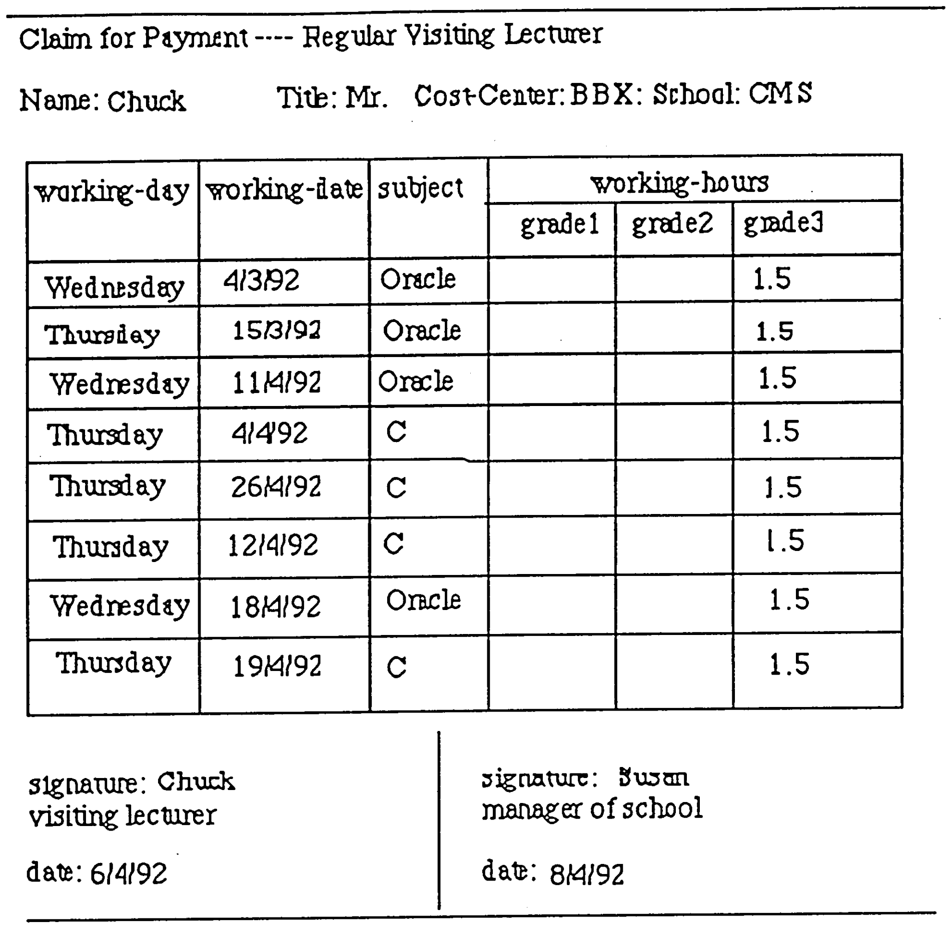
date: 6/4/92

signature: Susan
manager of school

date: 8/4/92

Figure A1.1

In this list, "instan181" corresponds to Figure A1.1, "inst181" corresponds to Figure A1.2 and "instan229" corresponds to Figure A1.3. Basically, the "list-form-instances" function searches through the type tree (Chapter 3) of the form instances, gives out the "subarea-path" of the schemas, and lists their values.



Claim for Payment ---- Regular Visiting Lecturer

Name: Chuck Title: Mr. Cost-Center: BBX: School: CMS

working-day	working-date	subject	working-hours		
			grade1	grade2	grade3
Wednesday	4/3/92	Oracle			1.5
Thursday	15/3/92	Oracle			1.5
Wednesday	11/4/92	Oracle			1.5
Thursday	4/4/92	C			1.5
Thursday	26/4/92	C			1.5
Thursday	12/4/92	C			1.5
Wednesday	18/4/92	Oracle			1.5
Thursday	19/4/92	C			1.5

signature: Chuck
visiting lecturer

date: 6/4/92

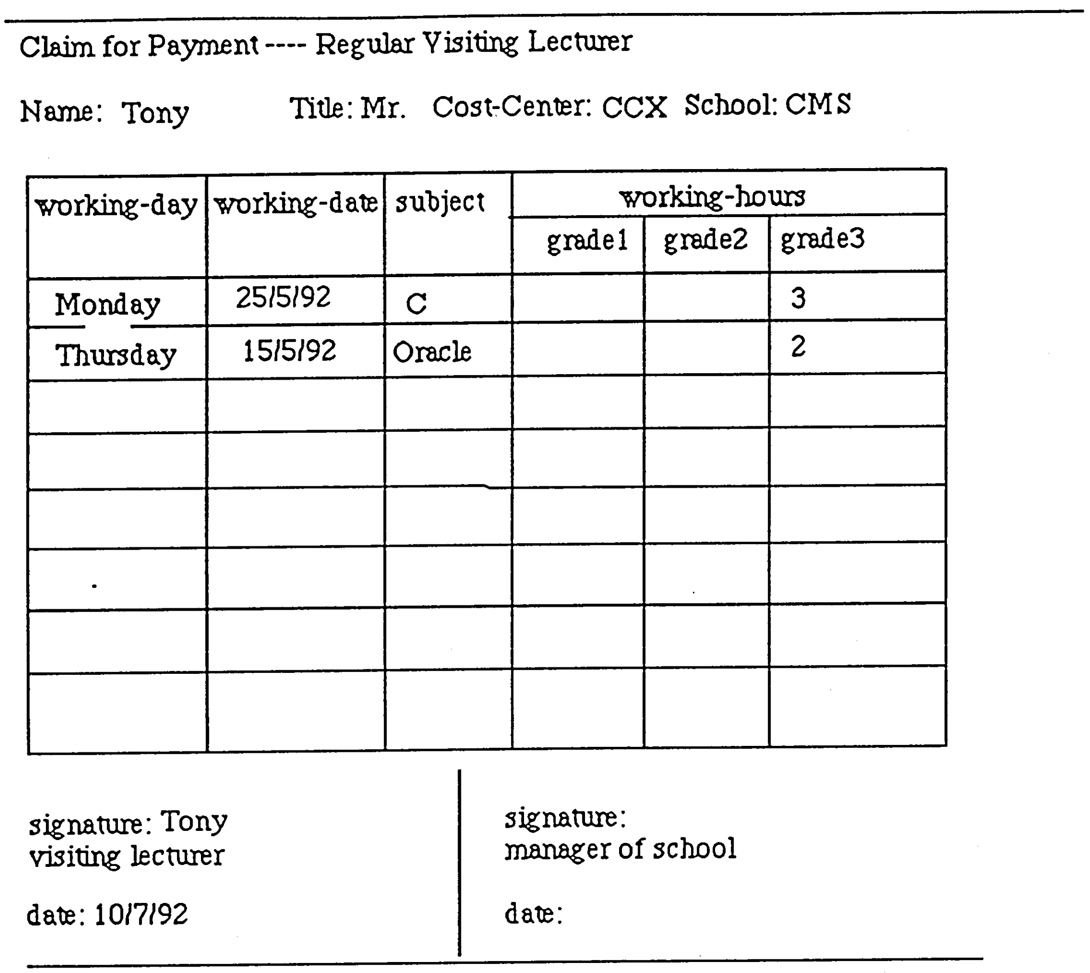
signature: Susan
manager of school

date: 8/4/92

Figure A1.2

We also can list all the ARVL form instances in the formbase by using the following function.

```
> (list-form-instances
          :forms '((?arvl (instance-of arvl)))
           :type 'arvl)
```



Claim for Payment ---- Regular Visiting Lecturer

Name: Tony Title: Mr. Cost-Center: CCX School: CMS

working-day	working-date	subject	working-hours		
			grade1	grade2	grade3
Monday	25/5/92	C			3
Thursday	15/5/92	Oracle			2

signature: Tony
visiting lecturer

date: 10/7/92

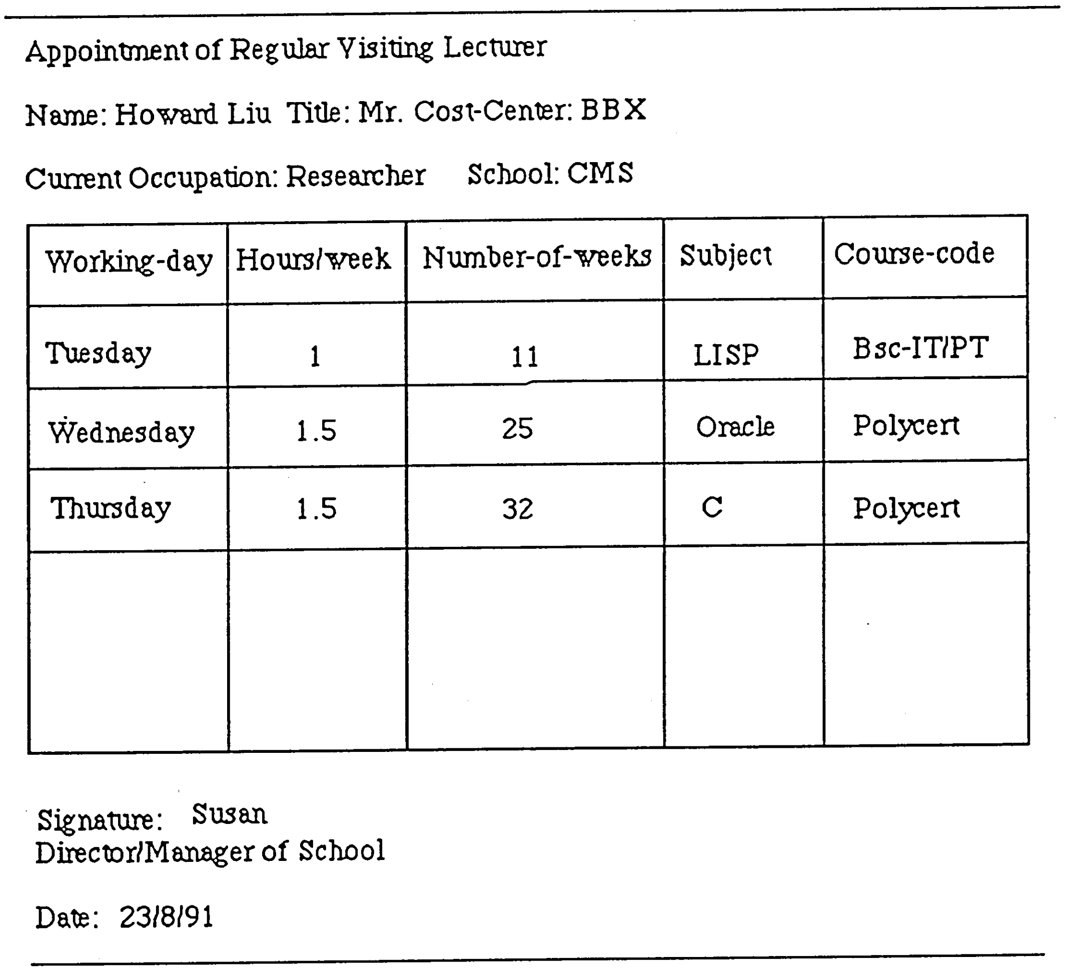
signature:
manager of school

date:

Figure A1.3

```
--------------- form instance: instan200
/ARVL
NAME: HOWARD TITLE: MR. COST-CENTRE: BBX CURRENT-
OCCUPATION: RESEARCH-ASSISTANT SCHOOL: CMS
/ARVL/TEACHING-COURSE
COURSE-CODE: POLYCERT SUBJECT: C NUM-OF-WEEK: 32
HOURS/PERW: 1.5 DAY: THURSDAY
/ARVL/TEACHING-COURSE
COURSE-CODE: BSC-IT/PT SUBJECT: LISP NUM-OF-WEEK: 11
HOURS/PERW: 1 DAY: THUESDAY
/ARVL/TEACHING-COURSE
COURSE-CODE: POLYCERT SUBJECT: ORACLE NUM-OF-WEEK: 25
HOURS/PERW: 1.5 DAY: WEDNESDAY
--------------- form instance: inst200
/ARVL
```

```
NAME: CHUCK TITLE: MR. COST-CENTRE: BBX CURRENT-
OCCUPATION: RESEARCH-ASSISTANT SCHOOL: CMS
/ARVL/TEACHING-COURSE
COURSE-CODE: BSC-IT/PT SUBJECT: LISP NUM-OF-WEEK: 11
HOURS/PERW: 1 DAY: THUESDAY
/ARVL/TEACHING-COURSE
COURSE-CODE: POLYCERT SUBJECT: ORACLE NUM-OF-WEEK: 25
HOURS/PERW: 1.5 DAY: WEDNESDAY
/ARVL/TEACHING-COURSE COURSE-CODE: POLYCERT SUBJECT: C
NUM-OF-WEEK: 32 HOURS/PERW: 1.5 DAY: THURSDAY
------------------------------------
NIL
>
```



Appointment of Regular Visiting Lecturer

Name: Howard Liu Title: Mr. Cost-Center: BBX

Current Occupation: Researcher School: CMS

Working-day	Hours/week	Number-of-weeks	Subject	Course-code
Tuesday	1	11	LISP	Bsc-IT/PT
Wednesday	1.5	25	Oracle	Polycert
Thursday	1.5	32	C	Polycert

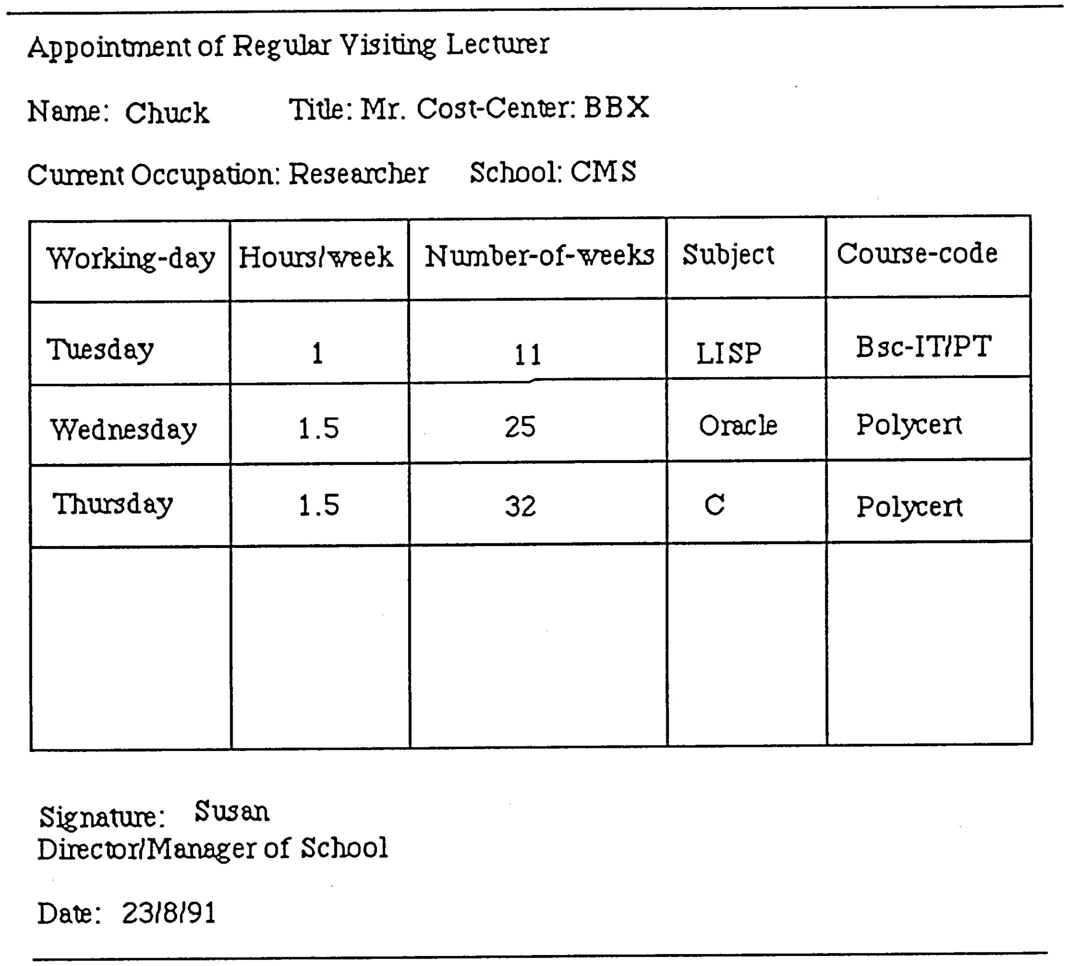
Signature: Susan
Director/Manager of School

Date: 23/8/91

Figure A1.4

In this list, "instan200" corresponds to Figure A1.4, "inst200" corresponds to Figure A1.5.

With these form instances in the formbase, shown as Figure A1.1, A1.2, A1.3, A1.4 and A1.5, examples of manipulations of the formbase system are demonstrated in the following sections. Queries to the formbase are demonstrated first, then is the predicates of the formbase, then is the formbase operations.



Appointment of Regular Visiting Lecturer

Name: Chuck Title: Mr. Cost-Center: BBX

Current Occupation: Researcher School: CMS

Working-day	Hours/week	Number-of-weeks	Subject	Course-code
Tuesday	1	11	LISP	Bsc-IT/PT
Wednesday	1.5	25	Oracle	Polycert
Thursday	1.5	32	C	Polycert

Signature: Susan
Director/Manager of School

Date: 23/8/91

Figure A1.5

1. Examples of Formbase Queries

```
>(formbase-query
            :forms '( (?rvl (instance-of RVL)
                            (school ?sch)
                            (time-tab (subject ?subject))
                            (name ?name))
                      (?arvl (instance-of ARVL)
                             (name ?name)
                            (school ?sch)
                            (teaching-course (subject ?subject)))))

Form RVL has following instances that satisfy the query: (instan181 inst181)

Form ARVL has following instances that satisfy the query: (instan200 inst200)

NIL
>

 >(formbase-query
            :forms '( (?rvl (instance-of RVL)
                            (school ?sch)
                            (time-tab (subject ?subject))
                            (name ?name))
                      (?arvl (instance-of ARVL)
                             (name ?name)
                            (school ?sch)
 (teaching-course (subject ?subject))))
    :type 'arvl)

 (ARVL (|instan200| |inst200|))

 >(formbase-query
          :forms '( (?rvl (instance-of RVL)
                          (school ?sch)
                         (time-tab (subject ?subject))
                         (name ?name))
                   (?arvl (instance-of ARL))
                          (name ?name)
                         (school ?sch)
                         (teaching-course (subject ?subject))))
         :type 'rvl
          :subarea-path (make-path :path-link 'time-tab))

 (TIME-TAB (|inst184| |inst188| |inst186| |inst192| |inst190| |inst198|
 |inst196| |inst194| |instan188| |instan190| |instan192| |inst an194|
 |instan184| |instan236| |instan198| |instan186|))
```

```
>(formbase-query
        :forms '( (?rvl (instance-of RVL)
                        (school ?sch)
                        (time-tab (subject ?subject))
                        (name ?name))
                  (?arvl (instance-of ARVL)
                         (name ?name)
                         (school ?sch)
                         (teaching-course (subject ?subject))))
        :type 'rvl
        :subarea-path (make-path :path-link 'time-tab)
        :attribute 'subject)

(C ORACLE)

>(formbase-query
          :forms '( (?rvl (instance-of RVL)
                          (school ?sch)
                          (time-tab (subject ?subject))
                          (name ?name))
                    (?arvl (instance-of ARVL)
                           (school ?sch)
                           (name ?name)
                           (teaching-course (subject ?subject))))
          :type 'rvl
          :subarea-path (make-path :path-link 'time-tab
                                   :selection '((working-date 15/3/92)))
          :attribute 'subject)

(C ORACLE)

>(formbase-query
          :forms '( (?rvl (instance-of RVL)
                          (school ?sch)
                          (time-tab (subject ?subject))
                          (name ?name))
                    (?arvl (instance-of ARVL)
                           (school ?sch)
                           (name ?name)
                           (teaching-course (subject ?subject))))
          :type 'rvl
          :subarea-path (make-path :path-link 'time-tab
                                   :selection '((subject ?subject)))
          :attribute 'subject)

(ORACLE C )
```

```
>(formbase-query
        :forms '( (?rvl (instance-of RVL)
                        (school ?sch)
                        (time-tab (subject ?subject))
                        (name ?name))
                  (?arvl (instance-of ARVL)
                         (school ?sch)
                         (name ?name)
                         (teaching-course (subject ?subject))))
        :type 'rvl
        :subarea-path (make-path :path-link 'time-tab
                                 :selection '((working-date 15/3/92))
                                 :sub-path (make-path
                                            :path-link 'working-hours ))
        :attribute 'grade3)

(1.5 )
```

2. Examples of Formbase Predicates

```
>(exists
      :forms '( (?rvl (instance-of RVL)
                      (school ?sch)
                      (time-tab (subject ?subject))
                      (name ?name))
                (?arvl (instance-of ARVL)
                       (name ?name)
                       (school ?sch)
                       (teaching-course (subject ?subject)))))
T
>(exists
      :forms '( (?rvl (instance-of RVL)
                      (school ?sch)
                      (time-tab (subject ?subject))
                      (name ?name))
                (?arvl (instance-of ARVL)
                       (name ?name)
                       (school ?sch)
                       (teaching-course (subject ?subject))))
      :type 'rvl :subarea-path (make-path :path-link 'time-tab))
T
>(exists
      :forms '( (?rvl (instance-of RVL)
                      (school ?sch)
```

```
                    (time-tab (subject ?subject))
                    (name ?name))
               (?arvl (instance-of ARVL)
                    (name ?name)
                    (school ?sch)
                    (teaching-course (subject ?subject))))
      :type 'rvl
      :subarea-path (make-path :path-link 'time-tab)
      :attribute 'subject)
T
>(exists
       :forms '( (?rvl (instance-of RVL)
                       (school ?sch)
                       (time-tab (subject ?subject))
                       (name ?name))
                 (?arvl (instance-of ARVL)
                       (school ?sch)
                       (name ?name)
                       (teaching-course (subject ?subject))))
       :type 'rvl
       :subarea-path (make-path :path-link 'time-tab
                                :selection '((working-day Wednesday)))
       :attribute 'subject)
T
```

3. Examples of Formbase Operations

```
 >(list-form-instances
       :forms '((?rvl (instance-of rvl)
                      (name Howard)))
       :type 'rvl)
--------------- form instance: instan181
/RVL
NAME: HOWARD TITLE: MR. SCHOOL: CMS COST-CENTRE: BBX
/RVL/MANAGER-SIGNATURE
DATE: 8/4/92 MANAGER-SIGNATURE: SUSAN
/RVL/LECTURE-SIGNATURE
DATE: 6/4/92 LECTURER-SIGNATURE: HOWARD
/RVL/TIME-TAB
WORKING-DATE: 4/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 26/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
```

```
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 15/3/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 4/3/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 19/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 18/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 12/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 11/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
-------------------------------------
NIL
>(Form-Op
        :forms '((?rvl (instance-of rvl)
                       (name Howard)))
        :type 'rvl
        :Op-type 'modify-value
        :attribute 'title
        :value 'Dr.)

>(list-form-instances
        :forms '((?rvl (instance-of rvl)
                       (name Howard)))
        :type 'rvl)
--------------- form instance: instan181
/RVL
TITLE: DR. NAME: HOWARD SCHOOL: CMS COST-CENTRE: BBX
/RVL/MANAGER-SIGNATURE
```

```
DATE: 8/4/92 MANAGER-SIGNATURE: SUSAN
/RVL/LECTURE-SIGNATURE
DATE: 6/4/92 LECTURER-SIGNATURE: HOWARD
/RVL/TIME-TAB
WORKING-DATE: 4/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 26/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 15/3/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 4/3/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 19/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 18/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 12/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 11/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
------------------------------------
NIL
>(Form-Op
        :forms '((?rvl (instance-of rvl)
                       (name Howard)))
        :type 'rvl
        :Op-type 'modify-value
        :subarea-path (make-path :path-link 'time-tab
                                        :selection '((working-date 15/3/92)))
        :attribute 'subject
        :value 'pascal)
>(list-form-instances
        forms '((?rvl (instance-of rvl)
                      (name Howard)))
        :type 'rvl)
--------------- form instance: instan181
/RVL
TITLE: DR. NAME: HOWARD SCHOOL: CMS COST-CENTRE: BBX
/RVL/MANAGER-SIGNATURE
DATE: 8/4/92 MANAGER-SIGNATURE: SUSAN
/RVL/LECTURE-SIGNATURE
DATE: 6/4/92 LECTURER-SIGNATURE: HOWARD
/RVL/TIME-TAB
WORKING-DATE: 4/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 26/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
SUBJECT: PASCAL WORKING-DATE: 15/3/92 WORKING-DAY:
THURSDAY
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 4/3/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 19/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 18/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 12/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
```

```
WORKING-DATE: 11/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
------------------------------------
NIL
>(Form-Op
        :forms '((?rvl (instance-of rvl)
                      (name Howard)
                      (lecture-signature (date 6/4/92))))
        :type 'rvl
        :subarea-path (make-path
                          :path-link 'time-tab
                          :selection '((working-date 15/3/92))
                          :sub-path (make-path
                                        :path-link 'working-hours))
        :Op-type 'modify-value
        :attribute 'grade3
        :value 3)
>(list-form-instances
        :forms '((?rvl (instance-of rvl)
                      (name Howard)))
        :type 'rvl)
--------------- form instance: instan181
/RVL
TITLE: DR. NAME: HOWARD SCHOOL: CMS COST-CENTRE: BBX
/RVL/MANAGER-SIGNATURE
DATE: 8/4/92 MANAGER-SIGNATURE: SUSAN
/RVL/LECTURE-SIGNATURE DATE: 6/4/92 LECTURER-SIGNATURE:
HOWARD
/RVL/TIME-TAB
WORKING-DATE: 4/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 26/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
SUBJECT: PASCAL WORKING-DATE: 15/3/92 WORKING-DAY:
THURSDAY
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 3 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 4/3/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
```

```
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 19/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 18/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 12/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 11/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
 -------------------------------------
NIL
 >(Form-Op
        :forms '((?rvl (instance-of rvl)
                       (name Howard)
                       (lecture-signature (date 6/4/92))))
        :type 'rvl
        :subarea-path (make-path
                          :path-link 'time-tab
                          :selection '((working-date 15/3/92))
                          :sub-path (make-path
                                        :path-link 'working-hours))
        :Op-type 'delete-value
        :attribute 'grade3)

>(list-form-instances
        :forms '((?rvl (instance-of rvl)
                       (name Howard)))
        :type 'rvl)
--------------- form instance: instan181
/RVL
TITLE: DR. NAME: HOWARD SCHOOL: CMS COST-CENTRE: BBX
/RVL/MANAGER-SIGNATURE
DATE: 8/4/92 MANAGER-SIGNATURE: SUSAN
/RVL/LECTURE-SIGNATURE
DATE: 6/4/92 LECTURER-SIGNATURE: HOWARD
```

```
/RVL/TIME-TAB
WORKING-DATE: 4/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 26/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
SUBJECT: PASCAL WORKING-DATE: 15/3/92 WORKING-DAY:
THURSDAY
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: NIL GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 4/3/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 19/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 18/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 12/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 11/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
------------------------------------
NIL
>(Form-Op
        :forms '((?rvl (instance-of rvl)
                       (name Howard)
                       (lecture-signature (date 6/4/92))))
        :type 'rvl
         :subarea-path (make-path
                          :path-link 'time-tab
                          :selection '((working-date 15/3/92)))
        :Op-type 'delete-subform)

>(list-form-instances
        :forms '((?rvl (instance-of rvl)
                       (name Howard)))
        :type 'rvl)
--------------- form instance: instan181
/RVL
TITLE: DR. NAME: HOWARD SCHOOL: CMS COST-CENTRE: BBX
/RVL/MANAGER-SIGNATURE
DATE: 8/4/92 MANAGER-SIGNATURE: SUSAN
/RVL/LECTURE-SIGNATURE
DATE: 6/4/92 LECTURER-SIGNATURE: HOWARD
/RVL/TIME-TAB WORKING-DATE: 4/4/92 WORKING-DAY: THURSDAY
SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 26/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 4/3/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 19/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 18/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 12/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 11/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
-------------------------------------
NIL
```

```
>(add-subform
        :forms '((?rvl (instance-of rvl)
                       (name howard)
                      (lecture-signature (date 6/4/92))))
        :type 'rvl
        :subarea-path (make-path :path-link 'time-tab)
        :value-list
            '((working-date 15/3/92)
              (working-day Wednesday)
             (subject Pascal)))

>(list-form-instances
        :forms '((?rvl (instance-of rvl)
                       (name Howard)))
        :type 'rvl)
--------------- form instance: instan181
/RVL
TITLE: DR. NAME: HOWARD SCHOOL: CMS COST-CENTRE: BBX
/RVL/MANAGER-SIGNATURE
DATE: 8/4/92 MANAGER-SIGNATURE: SUSAN
/RVL/LECTURE-SIGNATURE
DATE: 6/4/92 LECTURER-SIGNATURE: HOWARD
/RVL/TIME-TAB
WORKING-DATE: 4/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 15/3/92 WORKING-DAY: WEDNESDAY SUBJECT:
PASCAL
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: NIL GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 26/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 4/3/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 19/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
```

```
WORKING-DATE: 18/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 12/4/92 WORKING-DAY: THURSDAY SUBJECT: C
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
/RVL/TIME-TAB
WORKING-DATE: 11/4/92 WORKING-DAY: WEDNESDAY SUBJECT:
ORACLE
/RVL/TIME-TAB/WORKING-HOURS
GRADE3: 1.5 GRADE2: NIL GRADE1: NIL
-------------------------------------
NIL
>
```

Appendix 2.

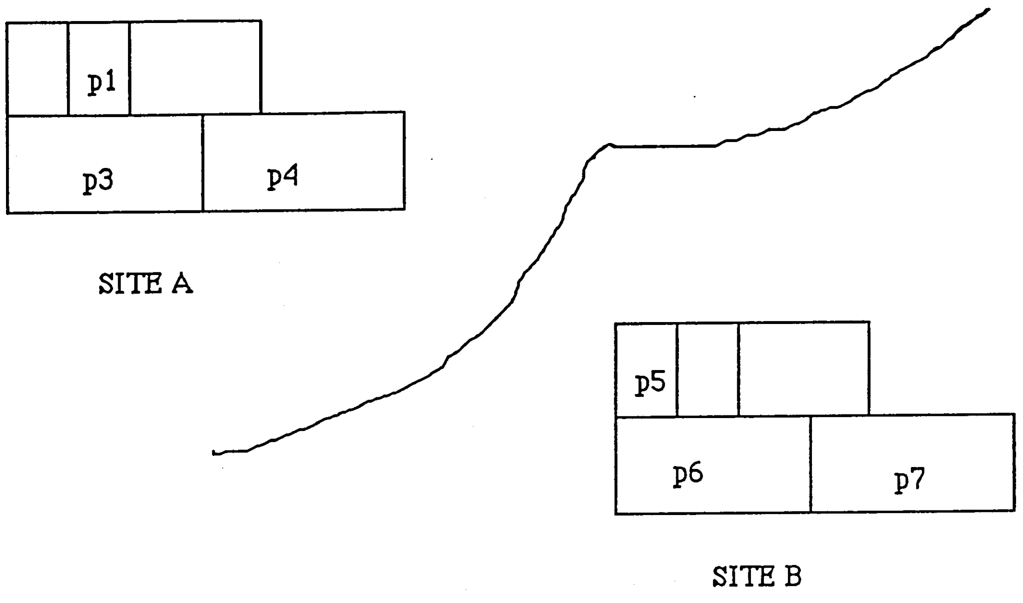
An Application of the Intelligent Form System: a Case of a Warehouse Management

A demonstration of task planning and agenda execution is introduced in this appendix. The case is about a warehouse management system. As introduced in section 5.4, the warehouse management system uses four types of forms. They are: table of space (TS, Figure 5.5), table of stored products (TSP, Figure 5.7), table of products (TP, Figure 5.6) and the goods transfer request form (GTRF, Figure 5.8). The demonstration is a formbase activity which transfers one product from one site to another site. Based on the activity knowledge and the situation inside the formbase, an agenda for fulfilling the transferring can be automatically generated by the Intelligent Form System. By executing the agenda, the effects of transferring the products will be recorded in the formbase.

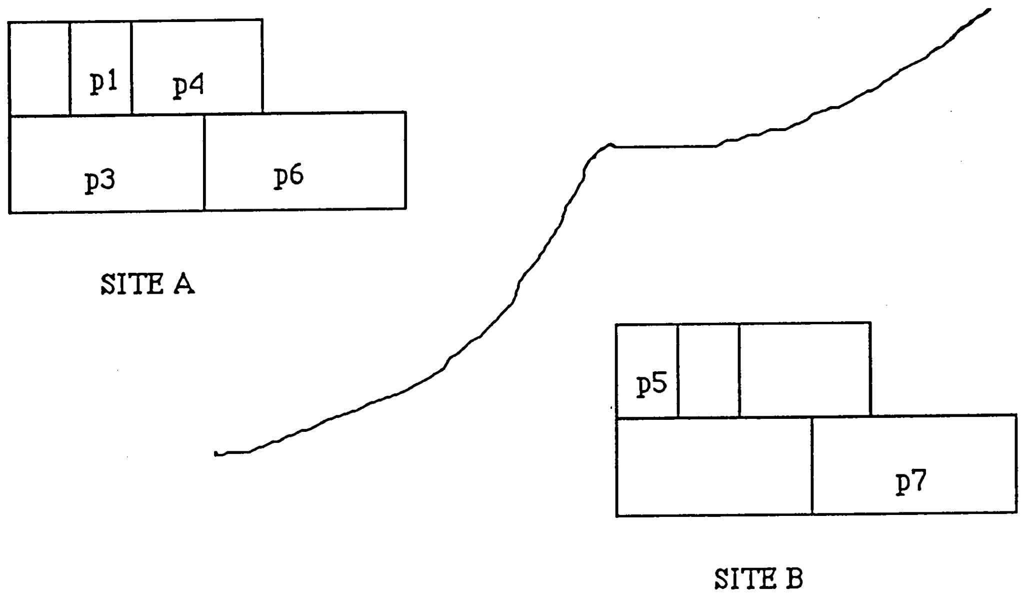
The initial situation of the warehouse is described by Figure A2.1 (a). The goal is to transfer product p6 from site B to site A. After the transferring, the situation is shown as Figure A2.1 (b).

By using IFS system, we first list the initial situation as the following:

```
> (list-form-instances
        :forms '((?ts (instance-of TS)
                      (location a)))
      :type 'ts)
```



(a) initial situation



(b) goal situation

Figure A2.1

```
--------------- form instance: instan243
/TS
LOCATION: A MANAGER: ROSS DATE: 1992
/TS/SPACE-TAB
SPACE-SIZE: 3 STATUS: OCCUPIED SPACE-ID: S4
/TS/SPACE-TAB
SPACE-SIZE: 1 STATUS: OCCUPIED SPACE-ID: S2
 /TS/SPACE-TAB
SPACE-SIZE: 1 STATUS: FREE SPACE-ID: S1
/TS/SPACE-TAB
SPACE-SIZE: 3 STATUS: OCCUPIED SPACE-ID: S5
/TS/SPACE-TAB
STATUS: FREE SPACE-SIZE: 2 SPACE-ID: S3
 ------------------------------------
NIL
> (list-form-instances
        :forms '((?tsp (instance-of TSP)
                       (site a)))
        :type 'tsp)

 --------------- form instance: instan255
/TSP
SITE: A MANAGER: ROSS TOTAL-SPACELEFT: NIL /TSP/WAREHOUSE-
TAB
SPACE-ID: S5 PRODUCT-TYPE: PT2 PRODUCT-ID: P4 /TSP/WAREHOUSE-
TAB
SPACE-ID: S4 PRODUCT-TYPE: PT2 PRODUCT-ID: P3 /TSP/WAREHOUSE-
TAB
SPACE-ID: S2 PRODUCT-TYPE: PT1 PRODUCT-ID: P1
 ------------------------------------
NIL
> (list-form-instances
        :forms '((?tsp (instance-of TSP)
                       (site b)))
        :type 'tsp)

 --------------- form instance: instan260
/TSP SITE: B MANAGER: BOB TOTAL-SPACELEFT: NIL
/TSP/WAREHOUSE-TAB
SPACE-ID: S5 PRODUCT-TYPE: PT3 PRODUCT-ID: P7 /TSP/WAREHOUSE-
TAB
SPACE-ID: S1 PRODUCT-TYPE: PT1 PRODUCT-ID: P5 /TSP/WAREHOUSE-
TAB
SPACE-ID: S4 PRODUCT-TYPE: PT3 PRODUCT-ID: P6
------------------------------------
```

Table of Space (TS)
Site: A Manager: Ross Date:

Space-id	Status	Space-size
s1	Free	1
s2	Occupied	1
s3	Free	2
s4	Occupied	3
s5	Occupied	3

Table of Space (TS)
Site: B Manager: Bob Date:

Space-id	Status	Space-size
s1	Occupied	1
s2	Free	1
s3	Free	2
s4	Occupied	3
s5	Occupied	3

(a)

Table of Stored Products (TSP)
Site: A Manager: Ross Total-space-left:

Product-id	Product-type	Space-id	Entry-date
p1	pt1	s2	
p3	pt2	s4	
p4	pt2	s5	

Table of Stored Products (TSP)
Site: B Manager: Bob Total-space-left:

Product-id	Product-type	Space-id	Entry-date
p5	pt1	s1	
p6	pt3	s4	
p7	pt3	s5	

(b)
Figure A2.2

```
NIL
> (list-form-instances
       :forms '((?ts (instance-of TS)
                     (location b)))
       :type 'ts)

--------------- form instance: instan249
/TS
LOCATION: B MANAGER: BOB DATE: 1992
/TS/SPACE-TAB
SPACE-SIZE: 1 STATUS: FREE SPACE-ID: S2
/TS/SPACE-TAB
SPACE-SIZE: 2 STATUS: FREE SPACE-ID: S3
/TS/SPACE-TAB
SPACE-SIZE: 3 STATUS: OCCUPIED SPACE-ID: S4
/TS/SPACE-TAB
SPACE-SIZE: 1 STATUS: OCCUPIED SPACE-ID: S1
/TS/SPACE-TAB
SPACE-SIZE: 3 STATUS: OCCUPIED SPACE-ID: S5
------------------------------------
NIL
> (list-form-instances
       :forms '((?tp (instance-of Tp)))
       :type 'tp)

 --------------- form instance: instan239
/TP
WAREHOUSE-NAME: MECHANICAL-HOUSE
/TP/PRODUCT-TAB
SPACE-REQUIRED: 3 PRODUCT-TYPE: PT3
/TP/PRODUCT-TAB
SPACE-REQUIRED: 1 PRODUCT-TYPE: PT1
/TP/PRODUCT-TAB
SPACE-REQUIRED: 2 PRODUCT-TYPE: PT2
------------------------------------
NIL
```

The initial situation which has been listed in the above corresponds to the the forms in Figure A2.2. The task of transferring product p6 from site B to site A can be represented by a predicate (transfer p6 b a). What the user needs to do is to call IFS by type "(gofs)" at lisp prompt as the following, then the system asks user to input task requirement. If the system is able to fulfil the task, an agenda will be generated.

```
> (gofs)

******** Goal Oriented Form System ********

  please input your task:(transfer p6 b a)

The activity agenda successfully generated

NIL
> (list-agenda)

score: 1        :goal (MOVE P4 A)
score: 2        :goal (FIND-SPACE 3 A)
score: 3        :goal (STORE P6 PT3 3 A)
score: 4        :goal (TRANSFER P6 B A)

NIL
>
```

After the agenda has been generated, it can be executed. After the execution, the contents of the forms should be changed so that information of the situation after the performance of the activity is correctly recorded.

```
> (run-agenda)

  **** Execution will delete the agenda,
do wish to save before this execution, y/n ? n

> (list-form-instances
        :forms '((?ts (instance-of TS)
                      (location a)))
        :type 'ts)

 --------------- form instance: instan243
/TS
LOCATION: A MANAGER: ROSS DATE: 1992
/TS/SPACE-TAB
STATUS: OCCUPIED SPACE-SIZE: 3 SPACE-ID: S4
/TS/SPACE-TAB
```

Table of Space (TS)
Site: A Manager: Ross Date:

Space-id	Status	Space-size
s1	Free	1
s2	Occupied	1
s3	Occupied	2
s4	Occupied	3
s5	Occupied	3

Table of Space (TS)
Site: B Manager: Bob Date:

Space-id	Status	Space-size
s1	Occupied	1
s2	Free	1
s3	Free	2
s4	Free	3
s5	Occupied	3

(a)

Table of Stored Products (TSP)
Site: A Manager: Ross Total-space-left:

Product-id	Product-type	Space-id	Entry-date
p1	pt1	s2	
p4	pt2	s3	
p3	pt2	s4	
p6	pt3	s5	

Table of Stored Products (TSP)
Site: B Manager: Bob Total-space-left:

Product-id	Product-type	Space-id	Entry-date
p5	pt1	s1	
p7	pt3	s5	

(b)
Figure A2.3

```
SPACE-SIZE: 1 STATUS: OCCUPIED SPACE-ID: S2
/TS/SPACE-TAB
SPACE-SIZE: 1 STATUS: FREE SPACE-ID: S1
/TS/SPACE-TAB
SPACE-SIZE: 3 STATUS: OCCUPIED SPACE-ID: S5
/TS/SPACE-TAB
STATUS: OCCUPIED SPACE-SIZE: 2 SPACE-ID: S3
------------------------------------
NIL

> (list-form-instances
        :forms '((?tsp (instance-of TSP)
                       (site a)))
        :type 'tsp)

 -------------- form instance: instan255
/TSP
SITE: A MANAGER: ROSS TOTAL-SPACELEFT: NIL /TSP/WAREHOUSE-
TAB
SPACE-ID: S4 PRODUCT-TYPE: PT2 PRODUCT-ID: P3 /TSP/WAREHOUSE-
TAB
SPACE-ID: S5 PRODUCT-TYPE: PT3 PRODUCT-ID: P6 /TSP/WAREHOUSE-
TAB
SPACE-ID: S3 PRODUCT-TYPE: PT2 PRODUCT-ID: P4 /TSP/WAREHOUSE-
TAB
SPACE-ID: S2 PRODUCT-TYPE: PT1 PRODUCT-ID: P1
------------------------------------
NIL
 > (list-form-instances
            :forms '((?tsp (instance-of TSP)
                           (site b)))
          :type 'tsp)

 -------------- form instance: instan260
/TSP
SITE: B MANAGER: BOB TOTAL-SPACELEFT: NIL /TSP/WAREHOUSE-
TAB
SPACE-ID: S5 PRODUCT-TYPE: PT3 PRODUCT-ID: P7 /TSP/WAREHOUSE-
TAB
SPACE-ID: S1 PRODUCT-TYPE: PT1 PRODUCT-ID: P5
------------------------------------
NIL
> (list-form-instances
        :forms '((?ts (instance-of TS)
                      (location b)))
        :type 'ts)
```

```
--------------- form instance: instan249
/TS
LOCATION: B MANAGER: BOB DATE: 1992
/TS/SPACE-TAB
SPACE-SIZE: 1 STATUS: FREE SPACE-ID: S2
 /TS/SPACE-TAB
SPACE-SIZE: 2 STATUS: FREE SPACE-ID: S3
/TS/SPACE-TAB
STATUS: FREE SPACE-SIZE: 3 SPACE-ID: S4
/TS/SPACE-TAB
SPACE-SIZE: 1 STATUS: OCCUPIED SPACE-ID: S1
/TS/SPACE-TAB
SPACE-SIZE: 3 STATUS: OCCUPIED SPACE-ID: S5
------------------------------------
NIL
>
```

The above vlaue list of the forms which are shown as Figure A2.3 corresponds to the situation after the transferring , as shown in Figure A2.1 (b).

Appendix 3.

Activity Representation for the Warehouse Management

The following is a list of the activity representation for the warehouse example demonstrated in appendix 2.

```
(make-activity
        :goal-pattern '(transfer $product-id $departure $destination)
        :holding-cond
              '(exists
                      :forms '((?gtrf (instance-of GTRF)
                                      (product-id $product-id)
                                      (present-site $departure)
                                      (destination-site $destination))))
        :assistant-activity
              '(let ((pro-type
                      (car (formbase-query
                              :forms '((?gtrf (instance-of GTRF)
                                              (product-id $product-id)
                                              (present-site $departure)
                                              (destination-site $destination)))
                              :type 'gtrf
                              :attribute 'product-type)))
                    (spa-size
                      (car (formbase-query
                              :forms '((?gtrf (instance-of GTRF)
                                              (product-id $product-id)
                                              (present-site $departure)
                                               (destination-site $destination)))
                              :type 'gtrf
                              :attribute 'space-required))))
                  (sub-goals 'store '$product-id pro-type spa-size '$destination))
        :effects
              '(let ((re-sp
                      (car (formbase-query  ;; find the removed space id
                              :forms '((?tsp (instance-of TSP)
                                              (site $departure)
                                              (warehouse-tab (product-id $product-id))))
                              :type 'tsp
                              :subarea-path (make-path :path-link 'warehouse-tab
                                              :selection '((product-id $product-id)))
                              :attribute 'space-id))))
```

```
                (Form-Op ;; remove transfer request
                    :forms '((?gtrf (instance-of GTRF)
                                    (product-id $product-id)
                                    (present-site $departure)
                                    (destination-site $destination)))
                    :Op-type 'delete-subform)
                (Form-Op ;; remove product info at departure site
                    :forms '((?tsp (instance-of TSP)
                                    (site $departure)
                                    (warehouse-tab (product-id $product-id))))
                    :type 'tsp
                    :subarea-path (make-path :path-link 'warehouse-tab
                                    :selection '((product-id $product-id)))
                    :Op-type 'delete-subform)
                (Form-Op ;; set the removed space free
                    :forms '((?ts (instance-of TS)
                                    (location $departure)))
                    :type 'TS
                    :subarea-path (make-path :path-link 'space-tab
                                    :selection `((space-id ,re-sp)))
                    :attribute 'Status
                    :Op-type 'modify-value
                    :value 'Free))
)


(make-activity
        :goal-pattern '(store $product-id $product-type $space-required $site)        :non-
assistant-cond
            '(some-bigger
                    :forms1 '((?ts (instance-of TS)
                                    (location $site)
                                    (space-tab (status Free))))
                    :type1 'ts
                    :subarea-path1 (make-path :path-link 'space-tab
                                            :selection '((status Free)))
                    :attribute1 'space-size
                    value2 '$space-required)
        :assistant-activity
            '(sub-goals 'find-space '$space-required '$site)
        :effects
            '(let ((sp-id (car (formbase-query
                                    :forms
                                     '((?ts (instance-of TS)
                                            (location $site)
                                            (space-tab (status Free))
                                            (space-tab (space-size $space-required))))
                                    :type 'ts
                                    :subarea-path (make-path
                                                    :path-link 'space-tab
                                                    :selection
                                                      '((status Free)
                                                        (space-size $space-required)))
                                    :attribute 'space-id))))
            (Form-Op
                 :forms `((?ts (instance-of TS)
                               (location $site)
                               (space-tab (space-id ,sp-id))))
                 :type 'ts
                 :subarea-path (make-path :path-link 'space-tab
                                          :selection `((space-id ,sp-id)))
                 :Op-type 'modify-value
                 :attribute 'status
                 :value 'Occupied)
             (add-subform
                 :forms '((?tsp (instance-of TSP)
                               (site $site)))
                 :type 'tsp
                 :subarea-path (make-path :path-link 'warehouse-tab)
                 :value-list
                         `((product-id $product-id)
                           (product-type $product-type)
                           (space-id ,sp-id)
                           (date nil))))
)


(make-activity
        :goal-pattern '(find-space $space-required $site)
        :holding-cond
              '(let ((sp-size (formbase-query
                              :forms '((?ts (instance-of ts)
                                            (location $site)))
                              :type 'ts
                              :subarea-path (make-path :path-link 'space-tab)
                              :attribute 'space-size)))
                (some-bigger
                     :value1 sp-size
                     :forms2 '((?ts (instance-of TS)
                                   (location $site)
                                   (space-tab (status Occupied))
                                   (space-tab (space-size $space-required))
                                   (space-tab (space-id ?space-id)))
                               (?tsp (instance-of TSP)
                                     (site $site)
                                     (warehouse-tab (space-id ?space-id))
                                     (warehouse-tab (product-type ?product-type)))
                               (?tp (instance-of TP)
                                    (PRODUCT-TAB (product-type ?product-type))))
                     :type2 'TP
                     :subarea-path2 (make-path :path-link 'PRODUCT-TAB)
                     :attribute2 'space-required ))
        :non-assistant-cond
              '(some-bigger
                     :forms1 '((?ts (instance-of TS)
                                   (location $site)
                                    (space-tab (status Free))))
                     :type1 'ts
```

```
                    :subarea-path1 (make-path :path-link 'space-tab
                                              :selection '((status Free)))
                    :attribute1 'space-size
                    :value2 '$space-required)
     :assistant-activity
          '(for (pro-id :in  (formbase-query
                               :forms
                                   '((?tsp (instance-of TSP)
                                           (site $site)
                                           (warehouse-tab (space-id ?space-id)))
                                    (?ts (instance-of TS)
                                           (location $site)
                                           (space-tab (space-size $space-required))
                                           (space-tab (space-id ?space-id))))
                               :type 'TSP
                               :subarea-path (make-path :path-link 'warehouse-tab
                                                     :selection '((space-id ?space-id)))
                               :attribute 'product-id))
             :do (if (some-bigger
                        :forms1
                           `((?ts (instance-of TS)
                                  (location $site)
                                  (space-tab (space-id ?space-id)))
                             (?tsp (instance-of TSP)
                                  (site $site)
                                  (warehouse-tab (space-id ?space-id))
                                  (warehouse-tab (product-id ,pro-id))))
                         :type1 'TS
                         :subarea-path1 (make-path :path-link 'space-tab)
                         :attribute1 'space-size
                         :forms2 `((?tsp (instance-of TSP)
                                         (site $site)
                                         (warehouse-tab (product-id ,pro-id))
                                         (warehouse-tab (product-type ?pro-type)))
                                   (?tp (instance-of TP)
                                        (PRODUCT-TAB (product-type ?pro-type))))
                         :type2 'tp
                         :subarea-path2 (make-path :path-link 'PRODUCT-TAB)
                         :attribute2 'space-required)
                      (sub-goals 'move pro-id '$site))) )


(make-activity
     :goal-pattern '(move $product-id $site)
     :non-assistant-cond ;; check if there is free space which is bigger than
                         ;; the space that the product requires.
        '(some-bigger
            :forms1 '((?ts (instance-of TS)
                           (location $site)
                           (space-tab (status Free))))
            :type1 'ts
            :subarea-path1 (make-path :path-link 'space-tab)
            :attribute1 'space-size
            :forms2 '((?tsp (instance-of TSP)
                    (site $site)
                    (warehouse-tab (product-id $product-id))
                    (warehouse-tab (product-type ?pro-type)))
               (?tp (instance-of TP)
                    (PRODUCT-TAB (product-type ?pro-type))))
        :type2 'TP
        :subarea-path2 (make-path :path-link 'PRODUCT-TAB)
        :attribute2 'space-required )
:assistant-activity
        '(let ((sp-size
               (car (formbase-query
                    :forms '((?tp (instance-of TP)
                                   (PRODUCT-TAB (product-type ?pro-type)))
                              (?tsp (instance-of TSP)
                                     (warehouse-tab (product-id $product-id))
                                     (warehouse-tab (product-type ?pro-type))))
                    :type 'TP
                    :subarea-path (make-path :path-link 'PRODUCT-TAB
                                     :selection '((product-type ?pro-type)))
                    :attribute 'space-required))))
           (sub-goals 'find-space sp-size '$site))
:effects
   '(let* ((old-sp
            (car (formbase-query ;; space the product stored
                  :forms
                    '((?tsp (instance-of TSP)
                            (site $site)))
                  :type 'tsp
                  :subarea-path (make-path :path-link 'warehouse-tab
                            :selection '((product-id $product-id)))
                  :attribute 'space-id)))
           (req-sp
            (car (formbase-query ;; the space-size the product required
                  :forms
                    '((?tsp (instance-of TSP)
                            (site $site)
                            (warehouse-tab (product-id $product-id))
                            (warehouse-tab (product-type ?pro-type)))
                      (?tp (instance-of TP)
                            (PRODUCT-TAB (product-type ?pro-type))))
                  :type 'TP
                  :subarea-path (make-path :path-link 'PRODUCT-TAB
                                     :selection '((product-type ?pro-type)))
                  :attribute 'space-required )))
          (new-sp (car  ;; space the product will move to
               (formbase-query
                    :forms `((?ts (instance-of TS)
                                   (location $site)
                                   (space-tab (status Free))
                                   (space-tab (space-size ,req-sp))))
                    :type 'ts
                    :subarea-path (make-path :path-link 'space-tab
                                     :selection `((space-size ,req-sp)))
                    :attribute 'space-id)))
        (pro-type
            (car (formbase-query ;; find out the type of the product
                    :forms '((?tsp (instance-of TSP)
                                    (site $site)
                                    (warehouse-tab (product-id $product-id))))
                    :type 'tsp
                    :subarea-path (make-path :path-link 'warehouse-tab
                                    :selection '((product-id $product-id)))
                    :attribute 'product-type))))
    (Form-Op  ;; free the old space
            :forms `((?ts (instance-of TS)
                            (location $site)
                            (space-tab (space-id ,old-sp))))
            :type 'ts
             :subarea-path (make-path
                            :path-link 'space-tab
                            :selection `((space-id ,old-sp)))
            :attribute 'status
            :Op-type 'modify-value
             :value 'Free)
    (Form-Op ;; record the occupation of the new space
            :forms `((?ts (instance-of TS)
                            (location $site)
                            (space-tab (space-id ,new-sp))))
            :type 'ts :subarea-path (make-path
                                      :path-link 'space-tab
                                       :selection `((space-id ,new-sp)))
            :attribute 'status
            :Op-type 'modify-value
            :value 'Occupied)
    (Form-Op ;; delete the old space information
            :forms `((?tsp (instance-of TSP)
                            (site $site)))
            :type 'tsp
            :Op-type 'delete-subform
            :subarea-path (make-path
                            :path-link 'warehouse-tab
                            :selection `((space-id ,old-sp))))
    (add-subform ;; record the new occupation information
            :forms `((?tsp (instance-of TSP)
                            (site $site)))
            :type 'tsp
            :subarea-path (make-path :path-link 'warehouse-tab)
            :value-list
                    `((product-id $product-id)
                      (product-type ,pro-type)
                      (space-id ,new-sp)
                      (date nil))))
)
```

References

Allen, J. F., and Koomen, J. A., "Planning using a temporal world model", in Proceedings of IJCAI-1983, pp. 741-747, 1983.

Allen, J. F. "Towards a General Theory of Action and Time", Artificial Intelligence 23 (1984) 123-154.

Allen, J., Hendler, J. and Tate, A. (eds) "Reading in Planning", Morgan Kaufmann Publishers Inc., 1990.

Banerjee, J., Chou, H. T., Garza, J. F., Kim, W., Woelk, D. and Ballou, N., Kim, H. J. "Data Model Issues for Object-Oriented Applications", ACM Transaction on Office Information Systems, Vol. 5, No. 1, January 1987, pp. 3-36.

Barber, G. "Supporting Organizational Problem Solving with a Work Station" ACM Transaction on OIS, Vol. 1, No. 1, January 1983.

Bond, A. and Gasser, L. "Reading in Distributed Artificial Intelligence", Morgan Kaufmann Publishers Inc., 1988.

Bracchi, G. and Pernic, B. "SOS: A Conceptual Model for Office Information Systems," Data Base, vol. 15, Winter 1984.

Bracchi, G. and Pernic, B."TRENDS IN OFFICE MODELLING," in Proceedings of the IFIP TC 8Working Conference, ed. R.A.HIRSCHHEIM, pp. 77-97, ELSEVIER SCIENCE PUBLISERS B.V, 1985.

Bruce, B. and Newman, D., "Interacting Plans", Cognitive Science, 2 (3), :195-233, 1978.

Cammarata, S., McArthur, D., and Steeb, R. "Strategies of Cooperation in Distributed Problem Solving" Proceeding of IJCAI, 1983, pp. 767-770.

Charniak, E. and McDermott, D. "Introduction to Artificial Intelligence" Addison-Wesley 1985

Chapman, D. "Planning for Conjunctive Goals" Artificial Intelligence 32 (1987) 333-377

Cattell, R.G.G., "Object Data Management: Object-Oriented and Extended Relational Database Systems", Addison-Wesley Publishing Company, 1991.

Checkland, P. "System Thinking, System Practice", John Wiley & Sons, Chichester.

Christodoulakis, M., Theodoridou, Ho, F., Papa, M., and Pathria, A., "Multimedia Document Presentation, Information Extraction, and Document Formation in MINOS: A Model and a System", ACM Tranction on Office Information Systems, Vol. 4, No. 4, October 1986.

Cohen, P. R., and Perrault, R. C. "Elements of a Plan-based Theory of Speech Acts", Cognitive Science, 3(3): 177-212, 1979.

Corkill, D.D., Gallagher, D.Q., and Johnson, P.M. "Achieving Flexibility, Efficiency, and Ganerality in Blackbroad Architectures", Proceeding of AAAI-1987, page 18-23.

Croft, B. and Lefkowitz, L. "Task Support In an Office System" ACM Transictions on Information System, Vol. 2, No. 3, July 1984, pp. 197-212.

Croft, B. and Lefkowitz, L. "A Goal-Based Representation of Office Work" IFIP Conference on Office Knowledge, 1988.

Currie, K., and Tate, A., "O-Plan - Control in the Open Planning Architecture", in Allen, J., Hendler, J. and Tate, A. (eds) "Reading in Planning", Morgan Kaufmann Publishers Inc., 1990.

Date, C.J., "An Introduction to Database Systems", Addison-Wesley Publishing company, Inc. 1990.

Dean, T. L., and McDermott, D. V., "Temporal Data Base Management", Artificial Intelligence 32 (1987), page 1-55, 1987.

Dean, T., Firby, R. J. and Miller, D. "Hierarchical Planning Involving Deadlines, Travel Times, and Resouces", Computational Inteligence Vol. 4, No. 4, pp. 381-398. [FORBIN]

Doyle, J. "A Truth Maintenance System" Artificial Intelligence 12 (1979) 231-272

Dreizen, H. M. and Chang, S. K. "Imprecise Schema: A Rational for Relations with Embedded Subreltions", ACM Transaction on Database Systems, Vol. 14, No. 4, December 1989, pp. 447-479.

Durfee, E. H., Lesser, V. R. and Corkill D. D. "Coherent Communication among Communicating Problem Solver" IEEE Transactions on Computers, C-36: 1275-1291, 1987.

Durfee, E. H., Lesser, V. R. and Corkill D. D. "Using Partial Global Plans to Coordinte distributed Prolem Solvers" in Proceeding of IJCAI-1987, page 875-883, 1987.

Ensor, J.R., and Gabbe, J.D. "Transactional Blackborads", International Journal for Artificial Intelligence in Engineering, 1(2): 80-84, 1986.

Ellis, C.G. "Information Control Nets: A Mathematical Model of Office Automation Flow," Proceedings of the 1979 Conference on Simulation, Measurement and Modellingof Computer System, 1979.

Feldman, J.A., and Sproull, R. F. "Decision Theory and Artificial Intelligence II: The Hungry Monkey", Cognitive Science 1, pp. 159-192, 1977.

Fikes, R.D. "REF-ARF: A System for Solving Problems stated as Procedures", Artificial Intelligence, 1(1970), 27-120.

Fikes, R.D., Nilsson, N.J. "STRIPS: a new approach to the application of theorem proving to problem solving", Artificial Intelligence, 2 (1971), 189-208.

Fikes, R.D., Hart, P.E. and Nilsson, N.J. "Learning and Executing Generalized Robot Plans", Artificial Intelligence, 3 (1972).

Fishman, D. H., Beech, D., Cate, H. P., Chow, E. C., Connors, T., Davis, J. W., Derret, N., Hoch, C. G., Kent, W., Lyngbaek, P., Mahbod, B., Neimat, M. A. and Shan, M. C. "Iris: An Object-Oriented Database Management System", ACM Transaction on Office Information Systems, Vol. 5, No. 1, January 1987, pp. 48-69.

Forgy, C. "A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", Artificial Intelligence, 19 (1982), 17-37.

Fox, M. S. "An Organizational View of Distributed Systems", IEEE Transaction on System, Man and Cybernetics, SMC-11: 70-80, 1981.

Fox, M. S. and Smith, S. F. "ISIS - a Knowledge-base System for Factory Scheduling", Expert System, Vol. 1, No. 1, July 1984.

Gasser, L. "Social conception of knowledge and action: DAI foundations and open systems semantics", 47 (1991) 107-138.

Georgeff, M. P. "Communication and Interaction in Multi-agent Planning" Proceeding of AAAI-1983, page 125-129, 1983.

Georgeff, M. P. "Theory of Action for MultiAgents Planning" Proceeding of AAAI-1984, page 121-125, 1984.

Georgeff, M. P. "The Representation of Events in MultiAgents Domain" Proceeding of AAAI-1986, page 70-75, 1986.

Georgeff, M. P. "Planning", Annual Reviews Computer Science, 2: 359-400, 1987.

Green, C., "Application of Theorem Proving to Problem Solving", Proceedings of IJCAI-1969, page 741-747, 1969.

Gibbs, S.J. "Conceptual Modelling and Office Information System", 1985

Greif, I. (ed.) "Computer Supported Cooperative Work: a book of readings", Morgan Kaufmann Publishers, Inc. 1988.

Hammer, M. and Howe, G.W. "A Very High Level Programming Language for Data Processing Applications," CACM, vol. 20, no. 11, 1977.

Hayes, P. J., "A Representation for Robot Plans", in Proceedings of IJCAI-1975, page 181-188, 1975.

Hayes, P.J. "The Logic of Frames" in "Frame Conceptions and Text Understanding" 41-61, edited by D. Metzing, Berlin: Walter de Gruyter and Co., 1979.

Hayes-Roth, B., and Hayes-Roth, F., "A Cognitive Model of Planning", Cognitive Science, 3(4) pp. 275-310, 1979.

Hayes-Roth, B. "A Blackbroad Architecture for Control", Artificial Intelligence, 26 (1985), page 251-321, 1985.

Hendler, J. A., "Integrating Marker-Passing and Problem Solving", in Allen, J., Hendler, J. and Tate, A. (eds) "Reading in Planning", Morgan Kaufmann Publishers Inc., 1990.

Hewitt, C. "Office Are Open Systems", ACM Transctions on OIS, Vol. 4, No. 3, July 1986, pp 271-287.

Hewitt, C. "Open Information Systems Semantics for Distributed Artificial Intelligence", Artificial Intelligence, 47 (1991) 79-106.

Hirschheim, R.A. "Understanding the Office: A Social-Analytic Perspective," ACM Transaction on Officce Information Systms, vol. 4, no. 4, pp. 331-344, October 1986.

Hornick, M. F. and Zdonik, S. B. "A Shared, Segmented Memory System for an Object-Oriented Database", ACM Transaction on Office Information Systems, Vol. 5, No. 1, January 1987, pp. 70-95.

de. Kleer, J "An Assumption-based TMS" Artificial Intelligence 24 (1984) 205-280

de. Kleer, J "Extending the ATMS" Artificial Intelligence 28 (1986) 163-196

de. Kleer, J "Problem Solving with the AMTS" 28 (1986) 197-224

Konsynski, B., Bracker, L. and Bracker, W. "A Model for Specification of Office Communications," IEEE Trans. Commun. COM-30,, vol. 1, pp. 27-36., Jan. 1982.

Korf, R. E., "Planning as Search: A Quantitative Approach", Artificial Intelligence, 33 (1987), pp. 65-88.

Kirsh, D. "Foundations of AI: the big issues", Artificial Intelligence, 47 (1991) 3-30.

Leao, L.V., and Talukdar, S.N. "COPS: A System for Constructing Multiple Blackbroads" International Journal for Artificial Intelligence in Engineering, 1(2): 70-79, 1986.

Leavitt, H., "Appiled Organizational Change in Industry", In: Handbook of Organizations, ed by J. March, Rand McNally, Chicago.

Lenat, D. B. "BEINGs: Knowledge as Interacting Expert" Proceeding of IJCAI, 1975, pp. 126-133.

Lesser, V. R., and Corkill, D. D. "Functionally Acurate, Cooperative Distributed Systems", IEEE Transaction on System, Man and Cybernetics, SMC-11 (1): 81-96, Januray, 1981.

Lifschitz, V. "On the Semantics of STRIPS" reprinted in Allen, Hendle and Austin (eds.) "Reading in Planning", Morgan Kaufman Publishers Inc., pp.523-530, 1990.

Liu, H., Draffan, I. and Poole, F. "A Representation for Office Form System", in Proceedings of 4th International Symposium On Artificial Intelligence, Mexico, November, 1991.

Liu, H., Draffan, I. and Poole, F. "A Goal Oriented Office Form System", ACM SIGOIS Bulletin, Vol 12, Number 2,3, pp. 123-128, November, 1991.

Lochovsky, F.H. "Managing Office Tasks." Proc. IEEE Computer Society Symposium on Office Automation, 27-9 April 1987, Gaithersburg MD, 206-16.

Lum, V.Y., Choy, D.M. and Shu, N.C. "OPAS: An Office Procedure Automation System," IBM System J, vol. 21, no. 3, p. 327, 1982.

Lockermann, P. C., Mayr, H. C., Weil, W. H., and Wohllere, W. H. "Data Abstractions for Data Systems" ACM Transaction on Database Systems, Vol. 4, No. 1, March 1979, pp. 60-75.

Malone, T. W. "ModellingCooridnation in Organizations and Markets", Management Science, 33 (10): 1317-1332, 1987.

Mazer, M.S. "Exploring the Use of Distributed Problem Solving in Office Support Systems" Proc. IEEE Computer Society Symposium on Office Automation, 27-9 April 1987, Gaithersburg MD, 217-25.

McDermott, D., "Planning and Acting", Cognitive Science, 2 (2), pp. 71-109, 1978.

McDermott, D., "A Temporal Logic for Reasoning About Processes and Plans", Cognitive Science 6, 1982, pp. 101-155.

Minsky, M. "A Framework for Representing Knowledge" in "Mind Design", 95-128, edited by J. Haugeland, Cambrige, 1981

Mogenstern, L. "Knowledge Preconditins for Actions and Plans", in Proceeding of IJCAI-1987, pages 867-874.

Moore, R. C., "A Formal Theory of Knowledge and Action", in Allen, J., Hendler, J. and Tate, A. (eds) "Reading in Planning", Morgan Kaufmann Publishers Inc., 1990.

Mylopoulos, J., Bernstein, P. A. and Wong, H-K. T. "A language Facility for Designing Database-Intensive Applications" ACM Transaction on Database Systems, Vol. 5, No. 2, June 1980, pp. 185-207.

Navathe, S. B., and Fry, J. P. "Restructuring for large Databases: Three Levels of Abstraction", ACM Transaction on Database Systems, Vol. 1, No. 2, june 1976, pp. 138-156.

Newell, A., and Simon, H.A., "GPS: A Program that Simulates Human Thought", in Allen, Henderler and Tate (eds.), "Reading in Planning", pp. 59-66, Morgen Kaufmann Publishers, Inc. 1990.

Nils J. Nilsson. "Principles of Artificial Intelligence" Springer-Verlag 1982

Peterson, J. "Petri Nets", Computing Surveys, Vol. 9, No. 3, September 1977.

Rosenschein, J. S. "Synchronization of Multi-Agent Plans", in Proceeding of AAAI-1982, pp. 115-119.

Rosenschein, J. S., Ginsburg, M., and Genesereth, M. R. "Cooperation without Communication", in Proceeding of AAAI-1986, pp. 51-57.

Rosenchein, J. S., "Plan Synthesis: A Logical Perspective", in Proceedings of IJCAI-1981, pp. 33-337, 1981.

Rosenschein, J. S., Ginsburg, M., and Genesereth, M. R. "Deals Among Rational Agents", in Proceeding of IJCAI-1985, page 91-99.

Roth, M. A., Korth, H. F. and Silberschatz, A. "Extended Algebra and Calculus for Nested Relational Databases", ACM Transaction on Database Systems, Vol. 13, No. 3, September 1988, pp. 389-471.

Sacerdoti, E.D., "Planning in a Hierarchy of Abstraction Spaces", Artificial Intelligence, 5 (1974), 115-135, 1974.

Sacerdoti, E.D. 1975 "The Non-linear Nature of Plans" Proc. of IJCAI-75. [NOAH]

Shoham, Y., and McDermott, D., "Problems in Formal Temporal Reasoning", Artificial Intelligence, 36 (1988), page 49 - 61, 1988.

Shipman, D. W. "The Functional Data Model and the Data Language DAPLEX", ACM Transaction on Database Systems, Vol. 6, No. 1, March 1981, pp. 140-173.

Smith, J. M., and Smith, D.C.P. "Database abstractions: Aggregation" Comm. ACM 20, 6 (June 1977), 405-413.

Smith, J. M., and Smith, D.C.P. "Database Abstractions: Aggregation and generalization" ACM Transaction on Database Systems, Vol. 2, No. 2, June 1977, 105-133.

Smith, R. G., and Davis, R. "Frameworks for Coo[eration in Distributed Problem solving" IEEE Transaction on System, Man and Cybernetics, SMC-11 (1): 61-70, 1981.

Steeb, R., Cammarata, S., Hayes-Roth, F. A., Thorndyke, P.W., and Wesson, R. B. "Distributed for Air Fleet Control" in "Reading in Distributed Artificial Intelligence" edited by Bond, A., and Gasser, L., Morgen Kaufmann Publishers, Inc., 1988.

Stefik, M. J. "Planning With Constraints" Artificial Intelligence 16 (1981) 111-140. [MOLGEN]

Stefik, M.J. "Planing and Meta-planning" Artificial Intelligence 16 (1981) 141-169 [MOLGEN]

Stonebraker, M., Anton, J. and Hanson, E. "Extending a Database System with Procedures", ACM Transaction on Database Systems, Vol. 12, No. 3, September 1987, pp. 350-376.

Stuart, C. J. "An Implementation of Multi-Agent Plan Synchronizer", in Proceeding of IJCAI-1985, page 1031-1033.

Suchman, L. A. "Plans and Situated Actions: the problem of human machine communication", Cambridge University Press, 1987.

Sussman, G.A. 1973 "The virtuous nature of bugs" , in Allen, J., Hendler, J. and Tate, A. (eds) "Reading in Planning", Morgan Kaufmann Publishers Inc., 1990.

Tate, A. 1976. "project Planning Using a Hierarchical Non-linear Planner". Dept. of Artificial Intelligence, Report 25, Edinburgh Univ. [NONLIN]

Tate. A 1977 "Generating Project Networks" Proc. of IJCAI-77 [NONLIN]

Tate. A 1984 "Goal Structure: Capturing the Intent of Plans" Proc. of ECAI-84 Pisa, Italy, 9, 1984 [NONLIN]

Tenney, R. R., and Sandell Jr., Nil R., "Strategies for Distributed Decisionmaking", IEEE Transation on Systems, Man and Cybernetics, SMC-11 (8): 527-538, 1981.

Tsichritzis, D. C. "Form Management," CACM, vol. 25, July,1982.

Tsichritzis, D., Fiume, E., Gibbs, S. and Nierstrasz, O. "KNO: KNowledge Acquisition, Dissemination, and Manipulation Objects", ACM Transaction on Office Informaiton Systems, Vol. 5, No. 1, January 1987, pp. 96-112.

Vere, S.A. "Planning in Time: Windows and Durations for Activities and Goals" IEEE Tranctions on Pattern Analysis and Machine Intelligence. Vol. PAMI-5, NO. 3, (1981) pp. 246-267. [DEVISER]

Vere, S.A. "Splicing Plans to Achieve Misordered Goals" Proc. of IJCAI-85. pp. 1061-1021. [DEVISER]

Waldinger, R., "Achieving Several Goals Simultaneously", in Allen, J., Hendler, J. and Tate, A. (eds) "Reading in Planning", Morgan Kaufmann Publishers Inc., 1990.

Wilensky, R., "A Model for Planning in Complex Situations", Cognition and Brian Theory, Vol. 4, No. 4, Fall, 1981.

Wilkins, D.E. "Domain Independent Planning: Representation and Plan Generation" Artificial Intelligence 22 (1984) [SIPE]

Wilkins, D.E. "Recovering from Execution Errors in SIPE" Computiational Intelligence 1 (1985). pp. 33-45 [SIPE]

Wilkins, D.E. "Practical Planning: extending the classical AI planning paradigm", Morgen Kaufmann Publishers Inc., 1988.

Woo, C.C. and Lochovsky, F.H. "Supporting Distributed Office Problem Solving in Organizations", ACM Transactions on OIS, Vol. 4, No. 3, 1986.

Zisman, M. D. "Use if Production System for Modelling Asynchronous,Concurrent Processes," Patten directed Inference Systems, Academic Press, 1978.

Zloof, M. M. "Office-By-Example: A Business Language that unifies Data and Word Processing and Electronic Mail," IBM Systems Jounal, vol. 21, Jan,1982.

Office Information System Modeling and
Artificial Intelligent Problem Solving

Conference for Computer Applications in the Social Sciences and Business

European Research Press, Portsmouth, September, 1991.

OFFICE SYSTEM MODELLING and PROBLEM SOLVING

Heyun Liu, Ian Draffan, Frank Poole

School of Computing and Management Sciences
Sheffield City Polytechnic
Hallamshire Business Park
100 Napier Street
Sheffield S11 8HD, U.K.
Email: Heyun@uk.ac.scp.cms

abstract
Modelling an office system is to identify a subsystem of the office system which can be hardened and merged with the social community of the office. In order to achieve this, it is necessary to study the connection between the routine office work and the decision making process, and the relation between the decision making process and the social activities of the office system. This paper presents a perspective which interprets an office system as a structure which is constructed by two levels of problem solving process. Comparing to the information processing perspective, this perspective is more open and dynamic; comparing to the open system perspective, it is more complete and close. A programming structure which supports this perspective is proposed.

1. Introduction

The perspective for an office system should promote the development of office system modelling. There are mainly two well-known perspectives for viewing an office system. One is the information processing perspective which views an office system as nothing but information processing [Price, 1979; Mokhoff, 1979]. The other is the open system perspective which views an office system as an open system [Carl, Hewitt, 1986]. Both of them have revealed certain natures of an office system. But the information processing persective is routine and inflexible, and the open system perspective is unstructured and loose.

1.1 Information Processing Perspective and its Failure

In the last decade, based on the persepctive that an office system is a well defined information processing system, the Office Information System modelling has generated many office models, such as SCOOP [Zisman, 1978], ICN [Ellis, C., 1979], FFM [Tsichritzis, D.C., 1982], OPAS [Lum, V.Y., 1982], and SOS [Bracchi, G., 1984]. These models can only model very routine office work. For example, SCOOP system [Zisman, 1978] is based on Petri Nets [Peterson, 1977]. ICN [Ellis, C., 1979] is based on the Information Control Net which is a modification of Petri Net. Both of them suppose that office work can be defined by a well structured network before the setting up of the system. Although later, the rule production system is used to model office models, such as SOS (Semantic Office System) [Bracchi, 1984], the demands for succinctly characterizing office work before the creation of the system is still very strong.

Therefore, these models only can describe the very routine phenomena of office work. This is impractical since very little office work can be rigidly defined in the office environment. Just as Barber pointed out in his paper [Barber, 1983], *"precisely because of its* *succinctness an office model suffers from two defects: first, it glosses over minor details that may be problematic or critical in practice; second, the reasons for the actions specified by a procedural description must be inferred"*. Therefore *"even routine tasks in offices encounter unexpected obstacles." This is because in the information processing approach "it is necessary to foresee the possible alternative courses of action when a procedural step cannot be performed."* But *"determining what the alternatives are is part of what office work is; all alternatives cannot be determined in advance."* Thus this approach *"is not a very useful style of work description because it needs to be augmented by the procedure's goal structure. When a procedure is augmented in this way, one can examine the procedure's goal structure in order to generate alternative steps when a step cannot be performed"* [Barber, 1983].

1.2 Open System Perspective and its Limitations

The unsatisfied developments of the information processing view enforce the researches in office automation to examine deeply the nature of the office system and its requirements. The investigation has revealed that an office information system is an open system which can be characterized by following features [Carl Hewitt, 1986]:

- Concurrency. To handle the simultaneous influx of information from many outside sources, the components of an office system must process information concurrently.
- Asynchrony. An office system functions in asynchrony to process the unpredictable input information. Besides, office systems may be impossible to be synchronized because of the physical distribution of its components.
- Decentralized control. Because of communications asynchrony and unreliability, a controlling agent could never have complete, uptodate information on the state of the office system. Therefore control must be distributed throughout the system so that local decisions can be made close to where they are needed.
- Inconsistent information. Information from outside and even the information from the different parts of the same office system may turn out to be inconsistent. Therefore decisions must be made by the components of an office system by considering whatever evidence is currently available.
- Arm-length relationships. The internal operation, organization, and state of one computational agent may be unknown and unavailable to another agent for reasons of privacy or outage of communications. Information should be passed by explicit communication between agents to conserve energy and maintain security. This ensures that each component inside an office system can be kept simple since it only needs to keep track of its own state and its interfaces to other agents.

The recognition of the above features of an office system, such as decentralized control, inconsistent information, and arm-length relations, has made it impossible for people to directly apply any theories and technologies developed for modelling any centrally controlled system to an office system. New foundations have to be developed for the modelling of an open system, such as an office system. An effort has been made by the scientists in the Artificial Intelligent circle of computer science. A new discipline called Distributed Artificial Intelligence (DAI) which aims to find out a solution for distributed problem solving in the multi-agents system [Alan H. Bond and Les Gasser, 1988] is emerging. The fundamental problems of DAI have been raised by the initial researches which are summarised by Gasser as follows [Gasser, 1991]:

- How to formulate, describe, decompose, and allocate problems and synthesize results among a group of intelligent agents.
- How to enable agents to communicate and interact; what communication languages or protocols to use, and what and when to communicate.
- How to ensure that agents act coherently in making decisions or taking action, accommodating the non-local effects of local decisions and avoiding harmful interactions.

- How to enable individual agents to represent and reason about the actions, plans and knowledge of other agents in order to coordinate with them; how to reason about the state of their coordinated process.
- How to recognize and reconcile disparate viewpoints and conflicting intentions among a collection of agents trying to coordinate their actions.

This list of problems actually represents another perspective of people viewing an open system (or an office system). It is based on a rather social observation of an organization.It has identified a set of problems which are fundamental for modelling an office system. However, it has been too strongly influenced by the distributed facts of an office system, so that it simply accepts the concurrent and asynchronous phenomena of an office system without analyzing them in per se. Therefore, the problems in above list does not imply a systematic approach for modelling office system. It is unstructured in terms of creating a computerized office system. It has ignored the fact that office work inside an organization is firstly inside a frame of an organizational problem solving process. It then secondly has its features no matter the office work is decomposition of tasks, or communication between agents, or reconciliation of disparate viewpoints.

Moreover, this perspective does not have a structure for analyzing the interactions between the partially well defined office activities and decision making processes. It can not set up connections between the decision making processes and the social community of the office system.

If modelling an office system is to identify a subsystem inside the office environment which can be hardened, and merged with the social community of the office system, it is necessary to study the connection between the routine office work and the decision making process, and the relation between decision making and the social activities. In the following, we first present a new perspective for an office system in Section 2, we then intrduce the technical base of this perspective ---- AI Planning system, in Section 3. In Section 4, we discuss two applications of the AI planning system in office system modelling. It is followed by a discussion of the interactions between human and computer problem solving process. Finally, a two dimensional problem solving structure for modelling office system is proposed in Section 6.

2. Problem Solving and Office System Modelling

An office is a very complex system. It can be viewed by many perspectives. Every perspective represents an interpretation of what an office model should be in the office environment. The information perspective interprets office model as a well defined activity network inside which the former always initiates the successors. The rationale of this viewpoint is due to the fact that in a well-defined sub-environment, once an office agent has chosen a method to fulfil the work, the office activities do have fixed relations. But this persective is criticized for not having a mechanism to reason which methods to choose. Therefore, the goal/task oriented approach looks more realistic [Barber, 1984].

However, the goal/task oriented approach can easily lead to the open system perspective since the context under which a goal is required is so complex and dynamic, and can not be represented easily. The open system perspective does identify a list of problems which are fundamental for modelling an office system. But it has no interpretation of the contexts under which the goals of the problems are proposed. Therefore, it is loose and unstructural. Needless to say, an organization does provide a context for every problem, the point is how it should be represented, since it may closely related to the solutions of problems.

There is another point which has not been addressed by the previous perspectives. That is the relationship between the office model and the office environment. This is specially important for the goal/task oriented approach, since the problems which concern how the initial goal is generated, and where the human interfere should happen have to be clear. In the previously developed modelling methodolgy, the concepts like goal, activity, problem solving process of the office model which we are going to set up, and these concepts of an office system itslef are not addressed separately. These concepts are discussed in a way which seems an office model will simply simulate an existing office system.

The perpsective we are going to present is based on the information processing perspective and the open system perspective, but it represents an interpretation of the working contexts of the office work, and an interpretation of the interaction between the office worker and the office model. The main assumptions and observations are:

Guide Line

Modelling an office system is to identify a subsystem of the office system which can be hardened and merged with the social community of the office system.

Human Machine Interaction

The computerized system should be considered as an intelligent system which has its own problem solving process, its own definition of goal/task, and activity. The interaction between an office worker and the computerized system is actually the interaction between the human problem solving process and the problem solving process of the computerized system.

Two Levels of Problem Solving

The office work inside an organization is organized by two levels of problem solving process. One is at the organizational level, the other is at the office agent level. The existence of a problem solving process at the office agent level is obvious. The existence of a problem solving process at the organizational level can be noticed through many observations. For example, when the business is small, a simple organization is enough to meet the requirements of the system functionality. But when business is enlarged, more complex structures have to be introduced to ensure the functionality and the efficiency of the organization. This expansion is a result of an application of the organizational problem solving process to the new requirements. We all aware the fact that a skilled manager would be able to solve his/her problems no matter in what type of organizations he/she is. This means that an organizational problem solving process for an organization should be able to be generalized. The interactions, or the balance of the organizational problem solving process and the agent problem solving process are very important for organization. Not only an organization is constructed based on these problem solving processes, but also the interactions, and communications of office agents inside office system are largely dependent on the interactions of the two problem solving processes. The failure of an organization is actually the failure of these problem solving processes in front of new requirements because of the organizational culture at that time.

It is a generalization of this observation that a computerized office system should be developed based on two levels of computer problem solving process. If an office model is developed based on problem solving processes, its functionality, flexibility, and applicability are all dependent on the ability of these problem solving processes. Moreover, the problems listed by Gasser [Gasser, 1991] should be considered based on these problems solving processes, and their relationships. It is these problem solving processes and their relationships that link the problems of an organization together. It should be a common base for the computer system to solve the organizational problems without falling into the ocean of technical details.

The problem solving process no matter at which level for a computer organization should not be a simple simulation of human organizational problem solving methodology. The human organizational problem solving process is too much related to the life activities of a human. For example, what is a proper size for people to work as a group? This is obviously related to the nature of the work. But it also related to many life activities such as holiday arrangements, possible sickness, even working time, etc. The size of a group will again influence the task decomposition, communication, concurrence and etc. Therefore, it is methodologically not right to simulate human organizational process.

3. The Problem Solving in AI Planning System

This section gives an introduction to the AI planning system. For more theoretical and technical discussion, please refer to [Allen, J., Hendler, J. and Tate, A., 1990].

3.1 The General Planner Theory

An AI planning system is charged with generating a plan which is one possible solution to a specific problem. The problem in the AI planning system is characterized by an initial situation and a goal situation description. The initial situation description tells the planning system the background of the problem in the world. The goal situation description tells the planning system what the world description should be after the plan has been executed. The plan generated will be composed out of activity schemas, which characterize activities and are provided to the planning system for each domain of applications. The activity schemas describe activities, or actions, in terms of their preconditions, steps, side-effects and goal. Each activity schema actually characterizes a class of possible events.

A plan is an organised collection of activities. A plan is said to be a solution to a given problem if the plan is applicable in the problem's initial situation, and if after plan execution, the goal is true. The plan is applicable if all the preconditions for the execution of this first operator hold in the initial situation. Repeated analysis can determine whether or not all the operators can be applied in the order specified by the plan. Repeated operator applications produce intermediate situation descriptions. If the goal is true at the end of this projection then the plan is a solution to the specified problem. So, the inputs to a typical AI planning system are a set of activity schemas and a problem which is characterized by an initial situation description and goal. The output from the planner is a plan which under projection satisfies the goal.

3.2 Plan Representation

To represent a planner, a network is used. The nodes of the network are activity descriptions. The arches of this graph are simple links which implies the order of the activities. The planning process is an expansion process of this network. This expansion is complex. Basically, the network is expanded by searching the activities whose goal can satisfy the goal or the subgoals the planner wants to achieve. This network has a start node and a finish node, the order from start node to the finish node represents the actual order of the activities.

3.3 Activity Schema

Using activity schema to describe the actual activities is very important in planning theory. It is based on the observation that all the activities no matter what level they are in an activity hierarchy can be described by a set of attributes. The set of attributes are goal, preconditions, side-effects, decompositions (steps), protections, and agents.

```
<activity>
{
   goal;
   preconditions;
   side-effects;
   decomposition: step1, step2 ...     ;; related subactivities
                                       ;;  for achieving the goal
   protections: (state1 time1 time2)
                (state2 time1 time2)
                        ... ...
```

```
                                        ;; states must be true
                                        ;; between timepoints
}
```

An activity schema actually describes a partially specified activity network. It is incompleteness, first it contains variables in its descriptions which potentially can unify a set of situation predicates and bring about a set of events. Second the order of the subactivity is not completely specified, since the order can only be enforced when protection intervals are violated.

3.4 Goal Achievement Procedure

Generally, the goal achieving process is divided into three steps.

1) The planner checks whether the goal is already true at the current node in the network. If it is already true then no more actions are needed, otherwise it carries on step 2.
2) In this step, the planner checks if there are some other nodes inside the network where the goal is true. If there are, it tries to link in one of the node to make the goal held at the current node.
3) If none of above steps succeeded, the planner searches the activity schemas which it has to find one which can achieve the goal, and expands the current network.

3.5 Outcomes of Planning

A planning system may have problems that it can never produce answer, because activity addition can make the plan grow arbitrary large, the search may never converge on a plan that necessarily solves the problem. Chapman [Chapman, 1987] discussed this problem. He pointed out that *"In fact, there are three possible outcomes (for the planning system): success, in which a plan is found; failure, when the planner has exhaustively searched the space of sequences of plan modification operations, and every branch fails; and nondetermination, when the plan grow larger and larger and more and more operations are applied to it, but it never converges to solve the problem."* Chapman proved a theorem for the correctness and completeness of Planner using the goal achievement procedure. That is, if planner, given a problem, terminates claiming a situation, the plan it produces does in fact solve the problem. If the planner returns signalling failure or does not halt, no solution exists.

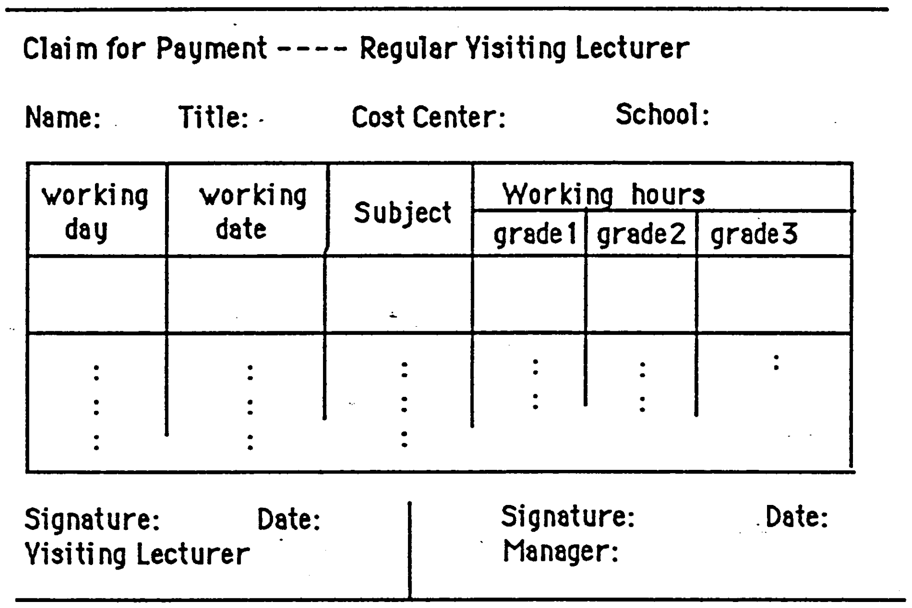
4. An Analysis of an Application of AI Planner to Office System

In this section, two office system modelling applications of the AI planning system are presented. These two applications are different in how to use the planning network. In the first application, the planning process is used to fulfil a transaction of office work. That is, whenever an office task is required, the system generates an activity network which is able to complete the task, then executes the network. The execution will update the data base which has information of the status of an office system, and meanwhile reset the planning network to 'nil'. In other words, it acts like a robot agent.

The second application is a program for scheduling a set of required tasks. The tasks will be fulfilled by a group of agents. The scheduling process is based on what functions an agent can perform for the fulfilling of a task, and what is the current loading of all the agents. The planning network will no longer just represent a transaction of work, but it provides information to the scheduling process. Specially, when large amount of tasks are concurrent, the existing planning network functions mostly as a set of constraints for the scheduling process.

4.1 A Goal-Oriented Office Form System

A goal oriented office form system is developed following the developing clue of the office information system modelling [Liu, H., 1991]. The planning process is able to directly manipulate office form instances. This achieves a very flexible link between the office information objects and the office work. For example, an office form shown as figure 1 can be represented as follows:



Claim for Payment ---- Regular Visiting Lecturer

Name: Title: Cost Center: School:

working day	working date	Subject	Working hours		
			grade1	grade2	grade3
:	:	:	:	:	:

Signature: Date:
Visiting Lecturer

Signature: Date:
Manager:

Figure 1.

```
<office-form-doc.>
{ (creator)
 (owner)
 (status)
 (create-date)
  ... ... }

<RVL-working-hours>
{  (is-a office-form-doc)
   (name)
   (title)
   (school)
   (cost-centre)
   {aggregation-of lecturer-signature
     (lecturer-signature)
     (date)}
   {aggregation-of  manager-signature
     (manager-signature)
     (date)}
   {association-of form-body
      (working-date)
      (working-day)
      (subject)
      {aggregation-of working-hours
(grade1)
(grade2)
(grade3) }}}}
```

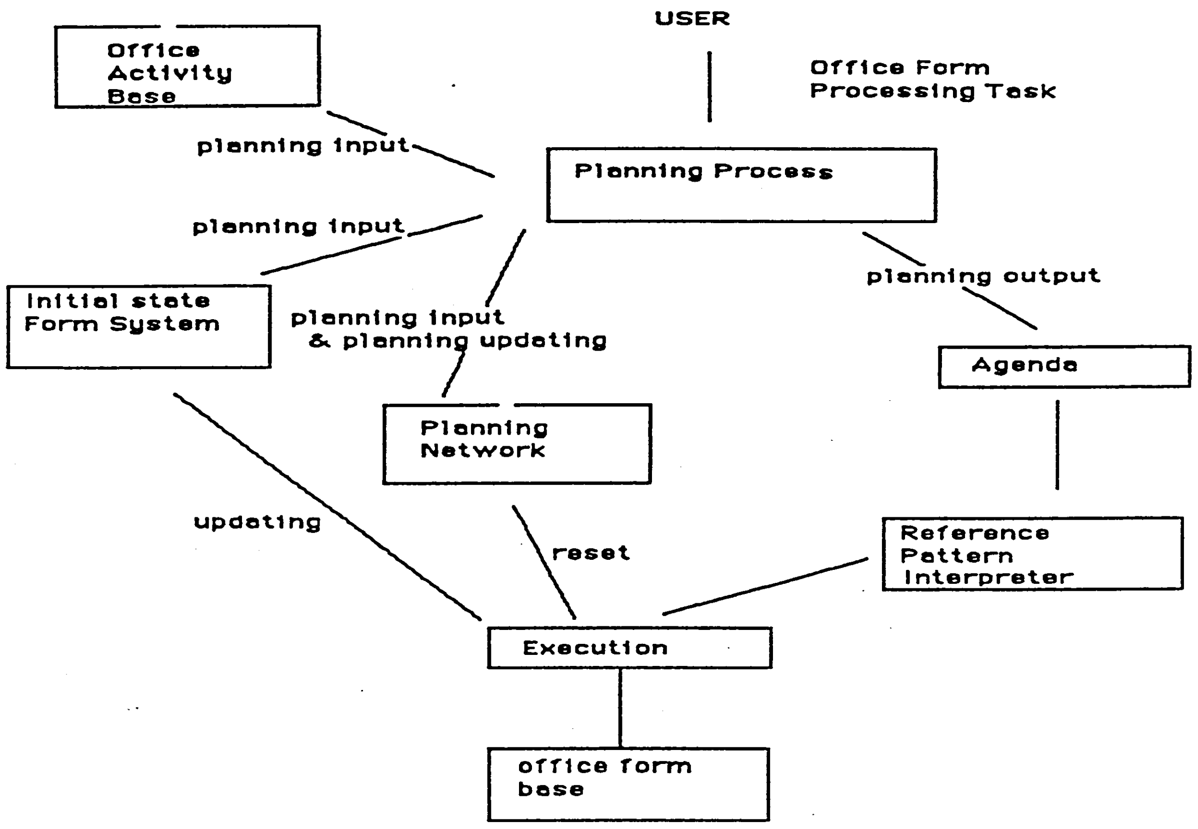


Figure 2.

The following is an example of an activity schema of this office form system. The situation is that after an office form system received an RVL form, it needs to verify it. The verify process can be expressed by the activity schema.

```
<verifying-RVL>
{
  office-forms: (?rvl &(instance-of RVL))
                (?arvl &(instance-of ARVL))
 preconditions: (received ?rvl)
 goal: (verified ?rvl)
 decomposition:
        goal-step signature-check
          = (exist (?rvl &(lecturer-sig (lecturer-sig))))
       goal-step contract-check
         = (and   (?rvl   &(is-a RVL-working-hours )
                         & (name ?name)
                         & (subject ?subject)
                         & (cost-center ?cc)
                         & (manager-sig (manager-sig Susan)))
              (?arvl &(is-a ARVL)
                        & (name ?name)
                       & (cost-centre ?cc)
                       & (manager-signed Susan)
```

& (@teaching-course (subject ?subject)))))

This activity schema states that when the system receives an instance of the RVL form, it first checks the signature and the contract. To check the signature it checks if the signature is exist. To check the contract, it will find an instance of the ARVL form, then check if the name, the subject, and the cost-centre in the RVL are the same as those in the ARVL. Both forms should be authorised by the same person. If all these are done, then the instance of the RVL form is verified.

The system structure of the goal oriented office form system is shown in figure 2. The planning network is used to represent a transaction of office work. When is task is required, the plan network for fulfilling the task is generated by the planning process based on knowledge of the situation and the activity schemas. The execution of the network will update the situation description data base, and reset the planning network to null. Therefore, this office form system functions like a robot agent. The planner required by this system needs not to be very complex, a SIPE [Wilkins, 1985] or NONLIN [Tate, A., 1977] style planner will be able to meet the requirements.

4.2 An Office Task Scheduling for Intelligent Agents

A scheduling system is simply a planning system. For scheduling, time constraints must be considered. So, it should be a DEVISER [Vere, S. A., 1981] style planner. But in order to schedule the agents, the activity schema is augmented with an agent slot which specifies who is able to perform the activity. The schema can be shown as follows:

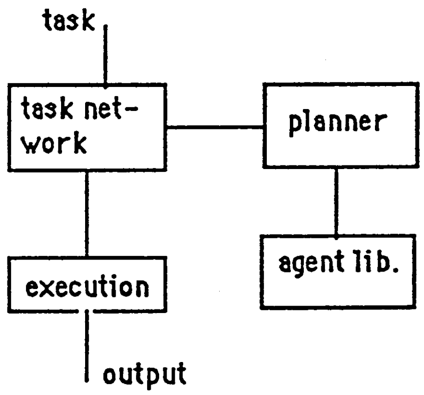


Figure 3.

```
<activity>
{
  goal;
  preconditions;
  side-effects;
     decomposition:  step1,  step2  ...            ;;  related  subactivities  for
achieving the goal
  protections: (state1 time1 time2)
               (state2 time1 time2)
                  ... ...                         ;; states must be true between
                                                  ;; timepoints
 time-window:
  agents;
}
```

In order to schedule office work, the generated planning network must be stored. It then functions as a set of constraints for the new emerging tasks. This cause a big change in the system structure.

The structure of the system is shown in figure 3. When a task is required, the system first enquires the task network that has already been stored in the system. If a sequence of activities for fulfilling this task is already there, the system just executes the sequence of the activities. Otherwise, the planner will be initiated to modify the task networks so that the new task is able to be fulfilled. The modification is based on the current task network and the agent library which has knowledge of the functions which the agents are able to perform.

5. Planning, Activity and Human Interfere

The AI planning system represents one interpretation of activities by the computer system. This interpretation is significant since it is clear in both organizational terms and programming terms. This interpretation can serve a base for us to developing the interaction of the computer problem solving process and the human problem solving process.

From the input of a task to the completion of the activity network, programmes are performed by a planner based on activity schema descriptions, the goal achieving process, and the control of searching as introduced in section 3. This is clear in programming terms. But what are the semantical meanings of these operations in the organizational terms. In another words, what is the semantical meaning of activity schema, and what is that of goal achieving process, and that of the control process?

The activity schemas of the planning system correspond to the well defined office activities inside an office system. For example knowledge of how to verify the RVL form as shown in Figure 1 is a well defined activity. The meaning of well defined is that the knowledge is independent from other activities. Once the preconditions of this activity is satisfied, this activity is applicable in the situation. The connection of the activities are set up by the situation description instead of by the activity description itself.

The goal achieving process and the search control is a further division of a problem solving process. The former refers to the ordering activity of a problem solving process, the later refers to the decision making activity of a problem solving process. Inside a planner, the goal achieving and the search controlling process recursively call each other to fulfil the goal. The goal achieving part can nicely solve most of the ordering problem, but the search part can make no decisions at all. It is actually the search strategy that simply replaces the decision making process. This probably is the reason why the applicability of the AI planners is limited to the problems which have mechanical natures. This implies a suggestion that in order to improve system the applicability of the problem solving process, human interfere happens exactly inside the search process.

Therefore, based on an AI planner, problem solving activity is divided into three levels. The well defined activity, the ordering activity, and the decision making activity. Human interactions should happen at the third level ---- the decision making level.

6. Two Dimensional Planning and Office Work Modelling

This section presents a new system structure for modelling office system. It is based on a concept called two dimensional planning. This concept is abstracted from many practical situation such as an organization. Inside an organization, all the work is at least inside two dimensions of problem solving. One dimension is the organizational problem solving process, the other is the problem solving process of the agent. It is the dynamic developments of the two dimensional problem solving processes that give an organization the flexibility and stability.

6.1 A New Frame for Modelling Office System

Office work is considered inside a two dimensional problem solving space in this new frame. One dimension represents the organizational problem solving, the other represents an agent problem solving. In the previous development of office system modelling, knowledge of the organization is usually modelled as static information. The disadvantages of this style of modelling is that there is no meta-problem solving process. Therefore, there are always some relationships which have to be well defined before the setting up of the system. This always means that many details of an organization can not be ignored. Without an organizational problem solving process to dynamically define the relationships between agents, their relationships and even the detailed reasons of the relationships have to be recorded. Again minor changes will cause problems. For example, if there is no organizational level problem solving process, in order to implement communication between the office agents, information has to be recorded for every possible agent to whom the communication might happen.

The organizational problem solving process also provides a base for the system to solve the problems listed by Gasser [Gasser, 1991]. Such as how to communicate, and how to reconcile disparate viewpoints, and so on. And the two dimensional problem solving should not influence human participation as discussed in the last section.

6.2 Interactions Between the two Dimensions

The most important problem for two dimensional problem solving is to find the join points of the two problem solving processes, and redefine the goal achieving process and control structures of both processes.

The interaction of the two problem solving processes should happen at the decision making activity level. This is similar to the problem solving inside an organization. When a manager needs to decide which office agent to be chosen for a specific job, he/she usually evaluates the possible consequences that they possibly will produce. And when an agent does not sure which other agent to contact for a problem, he/she usually consults a manager. These facts mean that inside a two dimensional problem solving space, both problem solving processes recursive call each whenever necessary.

7. Conclusion

Viewing office system as nothing but information processing is a misleading perspective, since it only can describe the superficial phenomena without being able to explain the reasons. Viewing office system as an open system, and tackling the corresponding problem solving strategies are actually trying to model office system by one level of problem solving processes. Since there is no meta-problem solving process, a large number of details have to be predefined before the setting up of the system, and the flexibility the system can achieve is limited.

The perspective which interprets that an organization is constructed by two levels of problem solving processes gives a very loose structure for the opened office system.

References

[Alan H.Bond and Les Gasser, 1988]
Alan H.Bond and Les Gasser, 1988, Reading in Distributed AI, Morgan Kaufmann Publishers, Inc.

[Allen, J., Hendler, J., and Tate, A., 1990]
Allen, J., Hendler, J., and Tate, A., 1990, Reading in Planning, Morgan Kaufmann Publishers, Inc.

[Barber, 1983] G. Barber, "Supporting Organizational Problem Solving with a Work Station" ACM Transaction on OIS, Vol. 1, No. 1, January 1983.

[Bracchi, 1984] G.Bracchi and B.Pernic, "SOS: A Conceptual Model for Office Information Systems," Data Base, vol. 15, Winter 1984.

[Bracchi, 1985] G.BRACCHI and B.PERNICAL, "TRENDS IN OFFICE MODELLING," in Proceedings of the IFIP TC 8 Working Conference, ed. R.A.HIRSCHHEIM, pp. 77-97, ELSEVIER SCIENCE PUBLISERS B.V, 1985.

[Chapman, 1987] David Chapman. "Planning for Conjunctive Goals" Artificial Intelligence 32 (1987) 333-377

[Croft & Lefkowitz, 1984]
Bruce Croft and Lawrence Lefkowitz "Task Support In an Office System" ACM Transitions on Information System, Vol. 2, No. 3, July 1984, pp. 197-212.

[Croft & Lefkowitz, 1988]
Bruce Croft and Lawrence Lefkowitz "A Goal-Based Representation of Office Work" IFIP Conference on Office Knowledge, 1988.

[Ellis, C 1979] C.G.Ellis "Information Control Nets: A Mathematical Model of Office Automation Flow," Proceedings of the 1979 Conference on Simulation, Measurement and Modeling of Computer System, 1979.

[FAOR, 1987] ESPRIT Project 56 Main Report, FAOR, 1987.

[Gasser, L., 1991] Gasser, L., 1991, Social conceptions of knowledge and action: DAI foundations and open systems semantics, Artificial Intelligence 47 (1991) 107 - 138

[Hewitt, C., 1986] Carl Hewitt, "Office Are Open Systems", ACM Transitions on OIS, Vol. 4, No. 3, July 1986, pp 271-287.

[Hewitt, C., 1991] Carl Hewitt, "Open Information Systems Semantics for Distributed Artificial Intelligence", 47 (1991) 79-106

[Hirschheim, 1986] R.A.Hirschheim, "Understanding the Office: A Social-Analytic Perspective," ACM Transaction on Office Information Systems, vol. 4, no. 4, pp. 331-344, October 1986.

[Lochovsky, 1986] Carson C. Woo and F. H. Lochovsky, "Supporting Distributed Office Problem Solving in Organizations", ACM Transactions on OIS, Vol. 4, No. 3, 1986.

[Lochovsky, 1987] F. H. Lochovsky. "Managing Office Tasks." Proc. IEEE

Computer Society Symposium on Office Automation, 27-9 April 1987, Gaithersburg MD, 206-16.

[Lum. V.Y., 1982] V.Y.Lum, D.M.Choy, and N.C.Shu, "OPAS: An Office Procedure Automation System," IBM System J, vol. 21, no. 3, p. 327, 1982.

[Mazer, 1987] M. S. Mazer. "Exploring the Use of Distributed Problem Solving in Office Support Systems" Proc. IEEE Computer Society Symposium on Office Automation, 27-9 April 1987, Gaithersburg MD, 217-25.

[Mokhoff, N., 1979] Mokhoff, N., 1979, Office Automation: A Chanllenge. IEEE Spectrum, No. 16.

[Peterson, 1977] James L. Peterson, "Petri Nets", Computing Surveys, Vol. 9, No. 3, September 1977.

[Price, S., 1979] Price, S., 1979, Intrducing the Electronic Office, NCC Publications, Manchester.

[Tate, 1977] Tate. A 1977 "Generating Project Networks" Proc. of IJCAI-77 [NONLIN]

[Tate, 1984] Tate. A 1984 "Goal Structure: Capturing the Intent of Plans" Proc. of ECAI-84 Pisa, Italy, 9, 1984 [NONLIN]

[Tsichritzis, D.C. 1982] D.C.Tsichritzis, "Form Management," CACM, vol. 25, July,1982.

[Vere, S.A., 1981] Vere, S.A. "Planning in Time: Windows and Durations for Activities and Goals" IEEE Tranactions on Pattern Analysis and Machine Intelligence. Vol. PAMI-5, NO. 3, (1981) pp. 246-267. [DEVISER]

[Vere, S.A., 1985] Vere, S.A. "Splicing Plans to Achieve Misordered Goals" Proc. of IJCAI-85. pp. 1061-1021. [DEVISER]

[Wilkins, 1984] Wilkins, D.E. "Domain Independent Planning: Representation and Plan Generation" Artificial Intelligence 22 (1984) [SIPE]

[Wilkins, 1985] Wilkins, D.E. "Recovering from Execution Errors in SIPE" Computiational Intelligence 1 (1985). pp. 33-45 [SIPE]

[Zisman, 1978] M.D.Zisman, "Use if Production System fir Modeling Asynchronous,Concurrent Processes," Patten directed Inference Systems, Academic Press, 1978.

A Representation for an Office Form System

The 4th International Symposium on Artificial Intelligence

Casncun, Mexico, November, 1991.
page of the Proceedings:336-342

A REPRESENTATION OF AN OFFICE FORM SYSTEM

Heyun Liu, Ian Draffan, Frank Poole

School of Computing and Management Sciences
Sheffield City Polytechnic
Hallamshire Business Park
Sheffield S11 8HD
United Kingdom

Abstract

An office form is a common office information object. It not only provides a structure for organizing office data, but also functions as an interface for office worker to access and manipulate them. To model office forms, one of an important issue is to handle the association of a form to its subforms, since in practice almost all the office forms have a subform which has a group of repetition of the same structure. Based on the frame system and pattern matching process in AI, this paper presents a pattern language which can solve this problem. By combining this pattern language with the AI style activity schema [Tate, 1984; Croft & Lefkotwiz, 1988], the activities upon office forms can be specified. This representation of the activities upon office forms is more complete and expressive.

1. Introduction

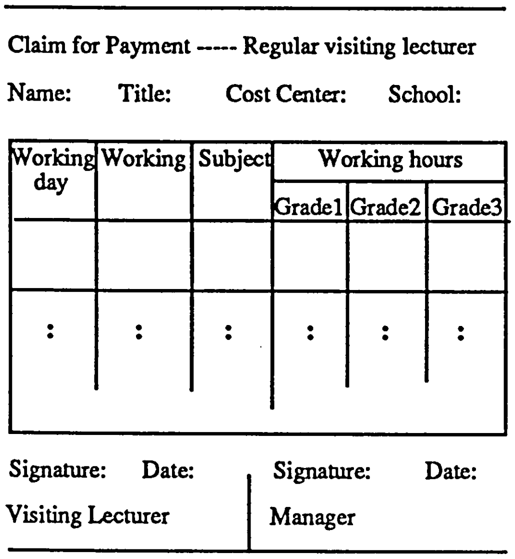
An office form is a common office information object. It not only provides a structure for organizing office data, but also functions as an interface for office workers to access and manipulate the data. The abstraction of an office form has led to an integrating office form concept [Tsichritzis, D., 1982] which means that computerized forms are not only the conceptual images of business paper forms, but are more general and elaborate so as to be able to represent any structural data and their templates in an office system. This concept has been used in many developed office models such as BDL [Hammer, M., 1977], Officetalk [Ellis, C., 1980], FFM [Tsichritzis, D., 1982], OPAS [Lum, V. Y., 1982], and SOS [Bracchi, 1984]. To model office forms, one of an important issue is to handle the association of a form to its subform, since in practice almost all the office forms have a subform which has a group of repetition of the same structure. For example in figure 1, Working day, Working date, Subject, and Working hours constitute a subform which has a group of instances. The data base oriented office form system, such as FFM, OPAS, cannot handle this association problem, since it is not normalized. The SOS model [Bracchi, 1984] proposes association abstraction, but does not have a way to handle it. Based on the frame system and pattern matching process in AI, this paper presents a pattern language which can solve this problem. It can model any office paper forms, and can also model a generalized abstract form which has any number of layers of subforms. This pattern language can be combined with the office activity schema proposed in POLYMER [Croft & Lefkowitz, 1988]. The office schema contains fields for describing the goal of the activity, steps for fulfilling the activity (decomposition), the temporal and causal relations among the activity's steps, preconditions and side effects of the activity, the agents responsible for performing the activity, and any additional constraints on the activity. Combining this office activity schema concept with the pattern language described in this paper, the representation of the activity schemes for the office forms can be specified.

2. A Pattern Language for an Office Form System

A pattern language for an office form system is presented in this section. The syntax of it can be found in Appendix A. The implementation of this language can be achieved by using the pattern matching process and the frame system in AI [Liu, H., 1990]. In the following, the definition of the type of an office form is developed first, then the reference patterns of the language is proposed, finally the operations and predicates for the forms are specified.

2.1. Definition of Form Type

To define the type of an office form, similar to the SOS [Bracchi, 1984] model, three types of abstractions are introduced. They are: generalization, aggregation, and association. Generalizations and aggregation respectively define superform and subform of types, and groups of elements of different fields. Associations specify groups of elements of the same type. For example, figure 1 is a claim for payment form, called RVL form, for an Regular Visiting Lecturer in an institute. It has three levels of information. The first level information is about the office form itself which usually people can not see from the surface. For example every office form has creators, owners, status, and etc. The second level is constituted by the less changeable information such as 'Name', 'Title', and 'School' in this example. The third level usually is the more detailed, and more changeable information. In this RVL form, for instance, 'Subject', 'Working Day', and etc. are obviously more changeable than the 'Name' of the lecturer. By using the


Claim for Payment ----- Regular visiting lecturer

Name: Title: Cost Center: School:

Working day	Working	Subject	Working hours		
			Grade1	Grade2	Grade3
:	:	:	:	:	:

Signature: Date: Visiting Lecturer | Signature: Date: Manager

Figure 1.

three abstractions, generalization, aggregation and association, the form can be represented as following.

```
<office-form-doc.>
{
 (creator)
 (owner)
 (status)
 (create-date)
 ... ...
}

<RVL-working-hours>
{
                ;; the first level information
   (is-a office-form-doc)
                ;; the second level information
   (name)
   (title)
   (school)
   (cost-centre)
   {aggregation-of  lecturer-signature
      (lecturer-signature)
     (date)
   }
   {aggregation-of   manager-signature
      (manager-signature)
     (date)
   }
                ;; the third level information
   {association-of form-body
     {aggregation-of
       (working-date)
       (working-day)
       (subject)
         {aggregation-of working-hours
          (grade1)
          (grade2)
          (grade3) }}}}
```

The syntax for defining office form type can be defined as follows:

```
<form-type> : :=
              <form-type-id> {<form-body-spec> }

<form-body-spec>
        : := Empty |
        : := <form-body-spec> <form-body-spec> |
       : := ( is-a <form-type-id> ) |
       : := { aggregation-of  <field-name>
                                   <field-body >} |
       : := { association-of  <form-type> }

<field-body>  : :=  Empty |
              : := <field-body>  <field-body> |
              : :=  <form-body-spec> |
              : := ( <attribute-name> )

     <form-name> : := symbol
<attribute-name>   : := symbol
    <field-name> : := symbol
```

Having this definition and given the office form concept is used as an integrating concept, almost all the structures of office information objects can be modelled. This data modelling method emphasises the semantic aspects of the information model.

2.2. Patterns for the Office Forms .

The names of forms, fields, and attributes can be used to refer to a particular set of instances of forms. For example, the pattern RVL-working-hours & (name H.Liu) may be considered to refer to all the instances of the RVL forms with a constraint 'name = H.Liu'. And the pattern RVL-working-hours & (name H.Liu) & (lecturer-signature (date "3/12/90")) refers to all the instances of the RVL form with constraint 'name = H.Liu' and the date of the signature is '3/12/90'. The reference pattern can get into more details. For example:

```
 RVL-working-hours & (name H.LIU)
      & (lecturer-signature (date "3/12/90"))
      & (@form-body & (subject "software tools")
      & (working-day "Monday"))
```

This is also a reference pattern upon the RVL-working-hours form. The symbol '@' is introduced to indicate that the symbol that follows it is a field defined by association abstraction. The field defined by association abstraction is regarded as a subform inside the form. So, the above pattern has two levels of constraints. The first level constraints are (name H.LIU) and (date "3/12/90"). The second level constraints are (subject "software tools") and (working-day "Monday"), and they are upon the repeated group (or subform) 'form-body'.

The syntax of reference pattern can be defined as follows:

```
<form-refer> : := <form-name> & <field-refer>

<field-refer> : := Empty |
              : := <field-refer> & <field-refer> |
              : := (<attribute-name> <value>) |
              : := (<field-name> (<field-refer>))
             : := (@<field-name> "&" <field-refer>)

<form-name> : := symbol
<field-name> : := symbol
 <value>  : := symbol | string | <variable> | ? | $?
 <variable> : := ?symbol
```

If a value is allowed to take a variable, the wild character '?', or the wild section character '$?', then the reference pattern is defined for this office form system. For example, if we have another form, called ARVL, in the institute for the Appointment of Regular Visiting Lecturer, as shown in figure 2.

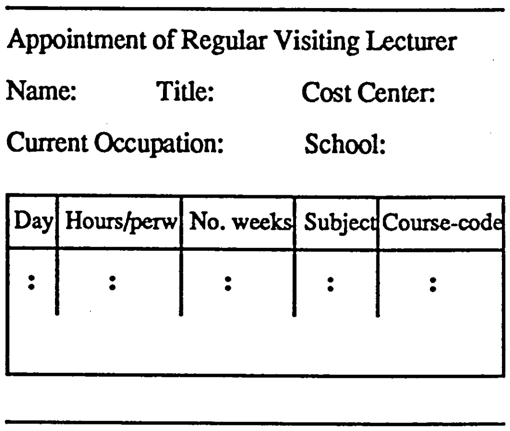
It can be defined as follows:

```
<ARVL>
{
   (is-a offi-form-doc)
   (name)
   (title)
   (school)
   (current-occupation)
   (cost-centre)
   (manager-signed)
   {association-of teaching-course
     {
       (day)
       (hours/perw)
       (number-of-weeks)
       (subject)
     }
   }
}
```

A matching pattern between an ARVL form and an RVL form can written as follow:

```
 (Forall (RVL-working-hours
                & (name ?name)
                & (subject ?subect)
                & (cost-centre ?cc)
               & (manager-sig (manager-sig Susan)))
        (ARVL & (name ?name)
                & (cost-centre ?cc)
                & (manager-signed Susan)
                & (teaching-course
                      (@subject &(?subject)))))
```

This pattern means for all the RVL form, there is a ARVL form which has the same name, same cost centre, same subject, and is authorised by the same person. Notice the matching between the variable '?subject' of the RVL form, and the variable '?subject' of the ARVL form. By employing the recursive pattern matching process, the system can set pattern restrictions by associating the values deep inside an associated subform



Appointment of Regular Visiting Lecturer

Name: Title: Cost Center:

Current Occupation: School:

Day	Hours/perw	No. weeks	Subject	Course-code
:	:	:	:	:

Figure 2.

with any other values.

2.3. Operations and Predicates

Like all the other data management system, the form document system support operations like create, delete, set, modify, and so on. Their syntax can be defined as following:

```
 (create <form-name> <form-type-id>)
        ;; create an instance of a form called 'form-name
          ;; whose type has already been defined
     (create <form-refer>)
          ;; create an instance of an associated subform
     (delete <form-refer>)
         ;; delete a value or an instance that the
         ;;  'form-refer' refers to
    (set <form-refer> <value>)
         ;; set the attribute referred by 'form-refer'
         ;;  to 'value'
   (modify <form-refer> <value>)
        ;; modify the attribute referred by 'form-refer'
        ;; to 'value'
```

All these operations are little different from the operations in a data base system. For example, if we have a form 'Julian-contract' which is an instance of the form type defined in figure 2, the operation (set (Julian-contract &(title)) Mr) will set the field 'title' to value 'Mr'. The operation (modify (Julian-contract &(title)) Dr) will modify the value of the field 'title' to a new value 'Dr'.

There is one thing needs a little more explanation. When an instance of an office form is set up, for example (create Julian-contract ARVL), the system always set up the it together with the instances of the subform which its association abstraction corresponds to. Usually association abstraction corresponds to an arbitrary number of repetition of the same structure. For example, inside an ARVL form (figure 2), the number of courses that the visiting lecturer is teaching is not fixed. So when the system creates an instance of the ARVL form, the teaching-course field should be created by an iteration procedure. Every time the system asks the user if there is any more to input. Only when the answer is negative, is the association abstraction created. More instances of the associated subform can be added in. For example, (create (Julian-contract (@teaching-course))) will create another instance of the associated subform 'teaching-course' for the form 'Julian-contract'.

The values and the status of the values of fields inside the office forms constitute part of the situation description for an office form system. Therefore having a set of predicates to express the status of values of a form are very important. Predicates can be defined for the status of a value of a form's attribute as following:

```
  (exist <form-refer>)
   ;; check if the value of the attribute existing
   (equal-to <form-refer> <value>)
  ;; if the value of the attribute is equal to the
  ;; value specified
  (equal-to <form-refer> <form-refer>)
 ;; if the value of the two attributes are equal
```

For example, the predicate

```
    (exist (Julian-contract
            &(name Julian
            &((@teaching-course
                &(subject "software tools"))))
```

returns true only when there exist at least one form 'Julian-contract' whose name is 'Julian', and the subject is 'software tools'. Following the same pattern, predicates like >, <, not-equal, and so on can be defined. With the help of these predicates, the system can define three kinds of integrity rules. The template constraints for the values of attribute fields of forms; the integrity rules for modification operations; and the semantic integrity rules to specify logical links among the fields.

3. A Goal Augmented Office Form Activity Representation

Having the above definition of office forms, the office activity schema defined by POLYMER [Croft & Lefkowitz, 1988] can be applied to model the office activities upon an office form. The syntax can be found in Appendix A. Following is an example of an office activity schema. The situation is that after an office worker receives an RVL-working-hours form, she needs to verify it. The verify process can be expressed by the activity schema shown as follows:

```
<verifying-RVL>
{
  preconditions  : ( (received ?rvl)
              & (instance-of ?rvl RVL-working-hours))
  goal           : (verified ?rvl)
  decomposition:
    goal-step signature-check
         = (exist ?rvl & (lecturer-sig (lecturer-sig ?)))
    goal-step contract-check
         =(Forall
      (?rvl & (name ?name)
             & (subject ?subect)
              & (cost-centre ?cc)
              & (manager-sig
                   (manager-sig Susan)))
       (?arvl & (name ?name)
              & (cost-center ?cc)
              & (manager-signed Susan)
              & ((@teaching-course
                   &(subject ?subject)))
       (instance-of ?arvl ARVL))
 Ordering     : signature-check before contract-check
 Agent        : Joan
}
```

This activity schema states that when agent Joan receives an instance of the RVL form (figure 1), she will first check the signature, then check the contract. To check the signature she just checks if the signature is there. To check the contract, she will find an instance of the ARVL form (figure 2), then checks if the name, the subject, and the cost-center in the RVL are the same as those in the ARVL. Both of the form should be authorised by the same person. If all these are done, then the instance of the RVL form is verified.

4. Discussions

By combining the pattern language proposed in this paper with the office activity schema proposed in [Croft & Lefkotwitz, 1988], a representation system for office forms can be specified (see Appendix A). Together with this pattern language, the activity schema upon the office form is more expressive and complete.

For example, following is an example of the office activity schema: [Croft & Lefkowitz, 1988]

```
ACTIVITY: Accept-or-Reject.Way1
Goal          : refereed(?paper)
 Preconditions: member (?paper, papers)
Decomposition: GOAL decision-reached =
                    exists (status (?paper))
 Control        : repeat decision-reached until
                    or (status (?paper, "accepted"),
                        status (?paper, "rejected"))
Agents        : ?editor = member (?editor, editors)
```

This activity schema aims to reflects the potential cyclic nature of journal editing. The activity attempts to achieve the goal of reaching a decision on the paper and indicating that this decision is either "accepted" or "rejected". If the decision is anything else, the task will continue. The activity schema has all the information. What is weak is the link between the activity and the data. More precisely, is the usage of predicates. Since the predicates can be freely chosen, the link between the activity and the data is loose, and open. And the modelling of the structure of office data runs a bigger risk of uncertainty. By using the pattern language described in this paper, many predicates can be designed as attributes of office forms, only is a small set of system predicates needed. This makes the system more complete, and stable. For example, in the above example an office form can be designed for the journal editing process, shown as figure 3.

Having defined this form, we can reconstruct the "accept-or-reject" activity schema as follow:

```
<accept-or -reject>
{
goal            : (exist (?paper &(decision)))
preconditions : (instance-of ?paper journal-editing)
decomposition: goal decision-reached=
                   (exist (?paper &(desion)))
control         : repeat decision-reached until
                 (or (?paper &(decision "accepted"))
                     (?paper &(decision "reject")))
}
```

This activity schema looks similar to the previous one, but its expressiveness is largely strengthened by the direct connection between the activity schema and the office form.

5. Conclusions

Based on the frame system and the pattern matching process in AI, a pattern language for an office form system can be defined. By combining this pattern language with the AI planner style activity schema, the activity schema for the office form system can be specified. Therefore, further research can concentrate on how to apply an AI problem solving process to office form system modelling.

References

G. Barber, "Supporting Organizational Problem Solving with a Work Station" ACM Transaction onOIS, Vol. 1, No. 1, January 1983

G.Bracchi and B.Pernici, "SOS: A Conceptual Model for Office Information Systems," Data Base, vol. 15, Winter 1984.

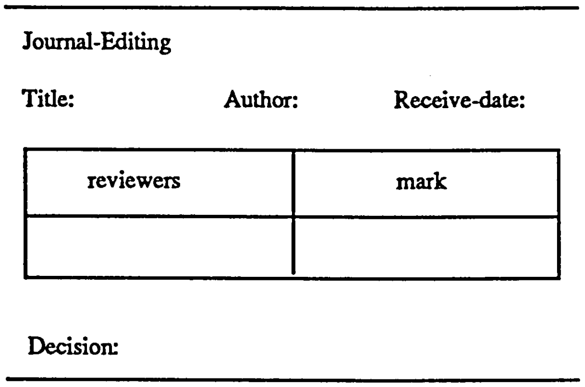
Bruce Croft and Lawrence Lefkowitz "A Goal-Based Representation of Office Work" IFIP Conference on Office Knowledge, 1988.

C.G.Ellis "Information Control Nets: A Mathematical Model of Office Automation Flow," Proceedings of the 1979 Conference on Simulation, Measurement and Modeling of Computer System, 1979.

M.Hammer and G.W.Howe, "A Very High Level Programming Language for Data Processing Applications," CACM, vol. 20, no. 11, 1977.

Liu, H., "Task Oriented Office Form System", Research report for transfer from Mphil to Ph.D, School of Computing and Managemant Sciences. Sheffield City Polytechnic, England, 1990.

V.Y.Lum, D.M.Choy, and N.C.Shu, "OPAS: An Office Procedure Automation System," IBM System J, vol. 21, no. 3, p. 327, 1982.



Journal-Editing

Title: Author: Receive-date:

reviewers	mark

Decision:

Figure 3.

Tate. A 1984 "Goal Structure: Capturing the Intent of Plans" Proc. of ECAI-84 Pisa, Italy, 9, 1984 [NONLIN]

D.C.Tsichritzis, "Form Management," CACM, vol. 25, July, 1982.

Appendix A
The BNF Grammar for GOOFS
(Goal Oriented Office Form System)

Office Form Documents Definition Language

```
<form-type> : := <form-type-id>
                  {<form-body-spec> }

<form-body-spec>
        : := Empty |
        : := <form-body-spec> <form-body-spec> |
        : := (is-a <form-type-id> ) |
        : := { aggregation-of <field-name>
                  <field-body> } |
        : := { association-of <form-type> }

<field-body> : := Empty |
             : := <field-body> <field-body> |
             : := <form-body-spec> |
             : := (<attribute-name>)

<form-name> : := symbol
<attribute-name> : := symbol
<field-name> : := symbol
```

Office Form Pattern Language

```
<form-refer> : := <form-name> & <field-refer>

<field-refer> : := Empty |
              : := <field-refer> & <field-refer> |
              : := (<attribute-name> <value>) |
              : := (<field-name> ( <field-refer> )) |
              : := ( @<field-name> & <field-refer> )


<form-name> : := symbol
 <field-name> : := symbol
<value>    : := symbol | ?symbol | string | ? | $?
```

Manipulations and Predicates on Office Forms

```
<form-operation>
        ::= (create <form-name> <form-type>) |
            (<operation1> <form-refer> <value>) |
            (<operation1> <form-refer> <form-refer>) |
            (<operation2> <form-refer>)

 <operation1> ::= set | modify
 <operation2> ::= delete

<form-predicate>
       ::=  (is-a <form-name> <form-type>) |
            (<predicate1> <form-refer><value>) |
            (<predicate1> <form-refer> <form-refer>) |
            (<predicate2> <form-refer>)

 <predicate1> ::= <> | = | > | < | >= |=<
 <predicate2> ::= exist
```

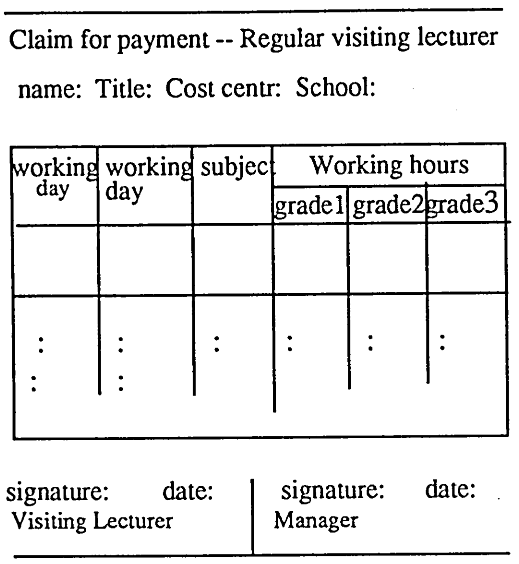
Office Form Activity Definition Language

```
 <office-activity-obj> : := <obj-name>
                           {
                            <goal>
                            <preconditions>
                            [ <side-effects> ]
                            [ <decomposition> ]
                            [ <plan-rationale> ]
                            [ <control> ]
                           }

  <goal>         : := <doc-wff>
 <doc-wff>       : := (<predicate> <form-refer>) |
                 : := (<predicate> <form-refer>
                                   <form-refer>)
 <predicate>     : := <system-prediate> |
                      <kb-predicate>
 <system-predicate> : := clear | existing |
                      equal | bigger | less |
                      sub-form | received | processed|
                      defined |member | instance-of
<preconditons>      : := <doc-wff>*
  <side-effects>    : := <doc-modification>*
 <doc-modification> : := (<doc-action>  <doc-wff>)
  <doc-action>      : := set | add | delete
<decomposition>     : := <activity-step>*
 <activity-step>   : := <step-spec>
                        [ done-by <agent-spec>]

 <step-spec> : := (goal <step-name> <subgoal-spec> )
|             (activity <step-name> <activity-spec>)|
                   (action <step-name> <action-spec> )

 <subgoal>        : := <doc-wff>
 <activity-spec>  : := <activity-name> |
                        (one-of  <activity-name>*)
 <action-spec>
     : := (<action-type1>  <form-refer> ) |
          (<action-type2>  <form-refer>  <form-refer> )
|         ( <action-type2>  <form-refer>  <value>)

 <action-type1>  : := set | delete | create | input | output
<action-type2>  : := + | - | * |\| modify
<plan-rationale>: := <enabling-relationship> |
                        <protection-interval>
<protection-interval>
           : := protect <states> [from <time-spec>]
                    to <time-spec>]

<states>    : := <doc-wff> | (<doc-wff>+)
<time-spec>: := <step-name> | before <step-name>|
                 after <step-name>

<enabling-relationship>
     : := <step-name> enables <enabling-step>

<control>  : := before <step-name> <step-name>+ |
           : := if <doc-wff> then <step-or-net>
                        [ else <step-or-net> ] |
            : := optional <step-or-net> |
            : := star <step-or-net>
            : := plus <step-or-net> |
            : := repeat <step-or-net> bounds
                  [ <iterate-when> <doc-wff>]
 <bounds> : := while <doc-wff> |
            : := until <doc-wff> |
            : := times count |
            : := for variable in value-list
 <step-or-net>  : := <step-name> | <plan-subnet>
 <plan-subnet> : := ( <step-name> to <step-name> )
<step-name> : := symbol
```

A Goal-Oriented Office Form System

Conference on Organizational Computing Systems
ACM SIGOIS Bulletin, Vol. 12, Number 2,3
pp. 123- 128, November, 1991.

abstractions are introduced. They are: generalization, ag-



Claim for payment -- Regular visiting lecturer

name: Title: Cost centr: School:

working day	working day	subjec	Working hours		
			grade1	grade2	grade3
: :	: :	:	:	:	:

signature: date: Visiting Lecturer	signature: date: Manager

Figure 1.

gregation, and association. Generalization and aggregation define superform and subform of types, and groups of elements of different fields respectively. Associations specify groups of elements of the same type. For example, figure 1 is a claim for payment form, called RVL form, for an Regular Visiting Lecturer in an institute. It has three levels of information. The first level information is about the office form itself which usually people can not see from the surface. For example every office form has creators, owners, status, and etc. The second level is constituted by the less changeable information such as 'Name', 'Title', and 'School' in this example. The third level usually is the more detailed, and more changeable information. For example, the 'Subject', and 'Working Day' attribute in the RVL form are obviously more changeable than the 'Name' of the lecturer. By using the three abstractions, generalization, aggregation and association, the form can be represented as following.

```
<office-form-doc.>
{ (creator)
  (owner)
  (status)
  (create-date)
  ... ... }

<RVL-working-hours>
{  (is-a office-form-doc)
   (name)
   (title)
   (school)
   (cost-centre)
   {aggregation-of  lecturer-signature
     (lecturer-signature)
     (date)}
   {aggregation-of  manager-signature
     (manager-signature)
     (date)}
   {association-of time-table
       (working-date)
       (working-day)
       (subject)
       {aggregation-of working-hours
         (grade1)
         (grade2)
         (grade3) }}}}
```

2.2. Patterns for the Office Forms

The names of forms, fields, and attributes are used to refer to a particular set of instances of forms. For example, the pattern (?rvl &(instance-of RVL) & (name H.Liu)) may be considered to refer to all the instances of the RVL forms with the constraint 'name = H.Liu'. And the pattern (?rvl &(instance-of RVL) & (name H.Liu) & (lecturer-signature (date "3/12/90"))) refers to all the instances of the RVL form with the constraints 'name = H.Liu' and the date of the signature is '3/12/90'. The reference patterns can get into more details. For example:

```
(?rvl  &(instance-of RVL)
       & (name H.LIU)
       & (lecturer-signature (date "3/12/90"))
       & (@form-body & (subject "software tools")
                     & (working-day "Monday"))
```

It is also a reference pattern upon the RVL form. The symbol '@' is introduced to indicate that the symbol that follows it is a field defined by association abstraction. The field defined by association abstraction is regarded as a subform inside the form. So, the above pattern has two levels of constraints. The first level constraints are (name H.LIU) and (date "3/12/90"). The second level constraints are (subject "software tools") and (working-day "Monday"), and they are upon the repeated group (or subform) 'time-table'.

If a value is allowed to take a variable, the wild character '?', or the wild section character '$?', then a pattern language is defined for an office form system. For example, if we have another form, called ARVL, in the institute for the Appointment of Regular Visiting Lecturer, as shown in figure 2. It can be defined as follows:

```
<ARVL>
{  (is-a offi-form-doc)
   (name)
   (title)
   (school)
   (current-occupation)
   (cost-centre)
   (manager-signed)
   {association-of teaching-course
```

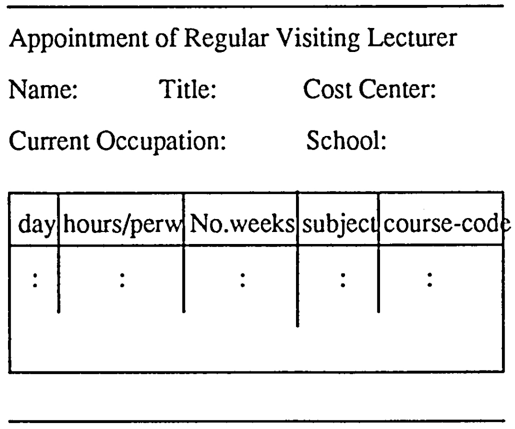
```
    {  (day)
       (hours/perw)
       (number-of-weeks)
       (subject) } } }
```

A matching pattern between an ARVL form and an RVL form can be written as follow:

```
 (and   (?rvl   &(instance-of  RVL-working-hours )
                & (name ?name)
                & (subject ?subject)
                & (cost-centre ?cc)
                & (manager-sig (manager-sig Susan)))
        (?arvl &(instance-of  ARVL)
               & (name ?name)
               & (cost-centre ?cc)
               & (manager-signed Susan)
               & (@teaching-course (subject ?subject)))))
```

This pattern means for all the RVL form, there is a ARVL form which has the same name, same cost centre, same subject, and is authorised by the same person.

The reference pattern for an office form is actually interpreted as an office type together with a set of constraints. The constraints can require an attribute of an office form has a special value, or that there is a special relationship existing


Appointment of Regular Visiting Lecturer

Name: Title: Cost Center:

Current Occupation: School:

day	hours/perw	No.weeks	subject	course-code
:	:	:	:	:

Figure 2.

between two attributes which could be inside the same form or inside different forms. For example the variable matching between the '?subject' of the RVL form, and the '?subject' of the ARVL form requires that an equivalent relation exists between these two attributes. More examples of the reference patterns can be found in [Liu, H., 1990].

There is another type of reference which does not use constraints. Once an instance of an office form is known, to refer to the value of a special attribute inside the form instance, the names of the attributes, subfields and subforms constitute a path to find the value. For example, if variable ?arvl has been bound to an instance of the ARVL type, the following reference pattern is a clear path to access the value of the subject attribute of the subform teaching-course of the form instance ?arvl.

```
        (?arvl &(instance-of ARVL)
               & (@teaching-course &(subject ?subject)))
```

2.3. Operations and Predicates

The basic operations for manipulating the office forms are little different from the operations in a data base system. For example, the operation (set (Julian-contract &(title)) Mr.) will set the field title to value 'Mr.'. The operation (modify (Julian-contract &(title)) Dr.) will modify the value of the field title to a new value 'Dr.'. There is one thing which needs to be mentioned. When an instance of an office form is set up, for example (create Julian-contract ARVL), the system always sets it up together with all the instances of the subform to which its association abstraction corresponds. Usually the association abstraction corresponds to an arbitrary number of repetitions of the same structure. For example, inside an ARVL form, the number of courses that the visiting lecturer teaches is not fixed. So when the system creates an instance of the ARVL form, the teaching-course field should be created by an iteration procedure. Every time the system asks the user if there is any more to input. Only when the answer is negative, is the association abstraction created. The type of an office form should be defined before we actually want to create an instance of the form.

The values and the status of the values of fields inside the office forms constitute part of the situation description for an office form system. Predicates can be defined for the status of a value of a form's attribute. For example, the predicate

```
        (exist (Julian-contract
                   &(name Julian)
                   &(@teaching-course
                              &(subject "software tools"))))
```

returns true only when there exist an ARVL form whose name is 'Julian', and the subject is 'software tools'. Following the same pattern, predicates like >, <, not-equal, and etc. can be defined.

3. The Internal Representation of an Office Form

To represent an office form, both the type definition of the office form and its instances are stored inside the system. To represent a type definition of an office form, a tree of schemes is used. The root schema represents the top level attributes of the form. The other schemes are related to the root schema through relation 'aggregation-of' or 'association-of'. Figure 3 and Figure 4 illustrate the representation of the definition of the RVL form (fig. 1). The RVL schema in figure 3 is the root schema of the RVL form. The others, lecturer-sig., manager-sig., time-table, and working-hours, are corresponding to the aggregation and association abstractions in the RVL definition. The relations of these schemes are then represented in

Figure 4, which illustrates the whole image of the representation of the RVL definition.

The instances of an office form are represented in the system by the same method. But the schemes which are used to represent an office form instance are the occurences of the instances of the schemes in the type representation of the office form. They are automatically generated and managed by the system, the names of these schemes are meaningless. For example figure 5 is a representation of an instance of the RVL form. The names of the schemes are generated by the system. In this example, G309 is an instance of RVL schema in the Figure 3, correspondingly G310, G311, G312, and G313 are instances of lecturer-sig., manager-sig., time-tab. and working-hours. Their connections constitute a representation of an instance of the RVL form. In the instance representation, the 'has-subfield' relation may connect a schema to a set of occurences of an association abstraction (for example, the time-table association abstraction in figure 5). This corresponds to the repetitions of a subform.

A reference pattern of an office form is a set of constraints upon all the instances of the type of the office form. An instance satisfies a reference pattern if all the constraints of different levels are satisfied by the instance. The process of the constraint satisfaction is guided by the stored type definition of the form.

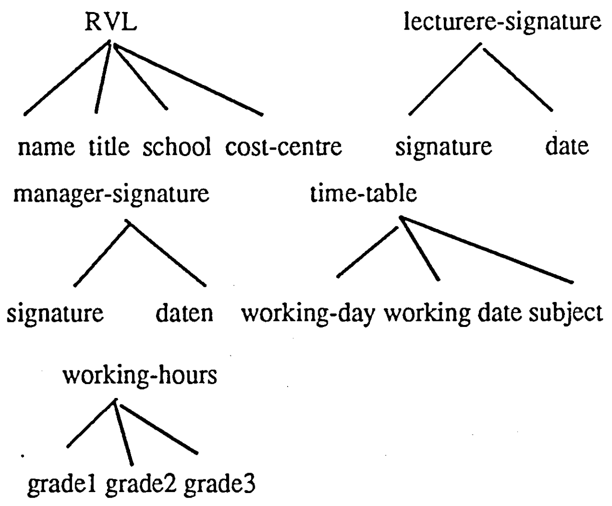


Figure 3.

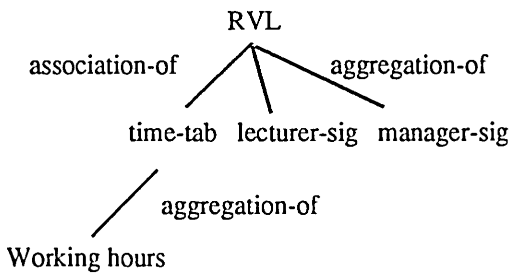


Figure 4.

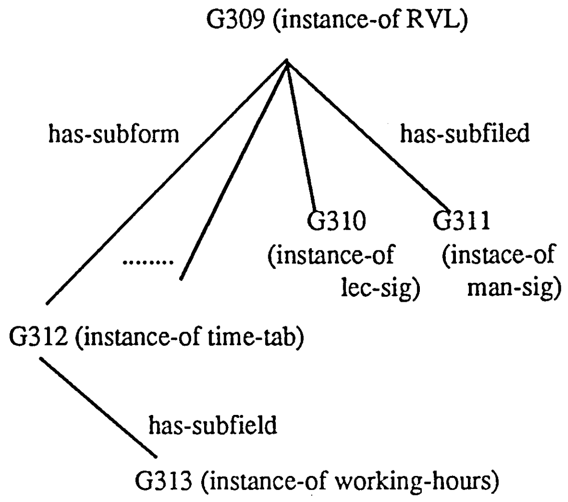

G309 (instance-of RVL)
has-subform
has-subfiled
....
G310 (instance-of lec-sig)
G311 (instace-of man-sig)
G312 (instance-of time-tab)
has-subfield
G313 (instance-of working-hours)

Figure 5.

4. A Planner for Office From System

A planner for the office form system is developed based on the office form pattern language. Some of the crucial problems of the planner are addressed in this section. To understand the discussion, knowledge of the AI planning system is required.

4.1 Representation of the Activity Schema

The activity schema concept is developed from the office activity schema of POLYMER[Croft, 1988] with the addition of the 'office-forms' slot which has constraints for the office forms used in the activity schema. The following is an example of an activity schema. The situation is that after an office form system receives an RVL form, it needs to verify it. The verification process can be expressed by the activity schema shown below.

```
<verifying-RVL>
{
  office-forms: (?rvl &(instance-of RVL))
                (?arvl &(instance-of ARVL))
 preconditions: (received ?rvl)
 goal: (verified ?rvl)
 decomposition:
     goal-step signature-check
           = (exist (?rvl &(lecturer-sig & (lecturer-sig))))
     goal-step contract-check
           = (and
              (?rvl  &(is-a RVL-working-hours )
                     & (name ?name)
              & (subject ?subject)
              & (cost-centre ?cc)
           & (manager-sig (manager-sig Susan)))
     (?arvl &(is-a ARVL)
              & (name ?name)
              & (cost-centre ?cc)
              & (manager-signed Susan)
    & ((@teaching-course (subject ?subject)))))}
```

This activity schema states that when the system receives an instance of the RVL form, it first checks the signature, then checks the contract. To check the signature it checks if the signature exists. To check the contract, it will find an instance of the ARVL form (fig. 2), then check whether the name, the subject, and the cost-centre in the RVL are the same as those in the ARVL form. Both forms should be authorised by the same person. If all these are done, then the instance of the RVL form is verified.

4.2 Representation of Planning Network

In common with other action ordering planners, the planning network is represented by the start, end, phantom, goal and activity nodes. The plan node (which contains basic information for goal node and activity node) contains information such as the value of the goal, preconditions, effects, pointers to predecessor, successor and parent nodes, etc.

Unlike the plan node of either NONLIN [A.Tate, 1984] or POLYMER[Croft, 1988], there is a field called 'office-forms'. The 'office-forms' field is actually a table whose every row contains a variable of a form, and its type, and a list of the constraints specified in the 'office-form' slot of the corresponding office form activity schema. The algorithm for finding all the instances of an office form which satisfies a set of constraints in the planning network is presented in section 4.4.

4.3 Form Effect Token

The effects of an activity in the planner is represented by either an ordinary predicate such as (verified ?rvl), or by a Form Effect Token. A Form Effect Token is a record which contains a sign field which is a (+) or a (-), an attribute reference which refers to an attribute of a form, and a value. If the sign is (+), it means the value of the attribute has been added into the system. Otherwise, if the sign is (-), it means the value of the attribute has retracted from the system. So, after the operation (set (Julian-contract &(title)) Dr.), the token (+, (Julian-contract &(title)), Dr.) should be added in the planning network.

With the help of the Form Effect Token, the TOME [Sacerdoti, 1975] and the GOST [A.Tate, 1984] structures are easy to set up to support the developing of the goal achieving process for the office form system.

4.4. Form Constraints

It is crucial for the planner to find the instances of an office form which satisfy a set of constraints in the planning network. In a reference pattern, like that shown below

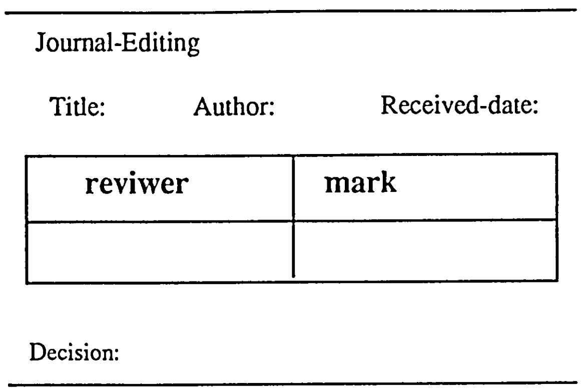
```
(and   (?rvl   &(is-a RVL-working-hours )
                 & (name ?name)
                 & (subject ?subject)
                 & (cost-centre ?cc)
                 & (manager-sig (manager-sig Susan)))
       (?arvl &(is-a ARVL)
                 & (name ?name)
                 & (cost-centre ?cc)
                 & (manager-signed Susan)
             & ((@teaching-course (subject ?subject)))))
```

the variables are used for the purpose of setting up constraints between the attributes. The attributes can be inside one form or inside different forms. To find the instances of a form which matches the reference pattern, the algorithm is divided into three steps. The first step is to abstract all the variables, and set up the constraint relationships based on the relations or the predicates upon these variables. The second step is to find all the instances of every type of the form which satisfies the constraints after the variables have been filtered. In above example, we need to find (?rvl &(instance-of RVL) &(manager-sig Susan)) and (?arvl &(instance-of ARVL) &(manager-sig Susan)) in this step. In the third step, the constraints set up in the first step are applied to the instances which are found in the second step, the results are the final set of instances that match the requirements.

The second step of the above algorithm is a searching process. Depending whether the form variable has been bound. If it has been bound, the search should start from where it was bound last time. If it is not bound, then the search should start from the initial office form base. It searches through the planning network. A constraint upon an attribute of a form is satisfied at a node, if it is either satisfied by the initially set of the instantiations of the variable of the office form (for unbound form variable, the initial set of instantiation includes all the instances of the form type), or satisfied by a predecessor node of this node, and no Form Effect Token been added in before this node, which has a negative effect for the same attribute.

5. Discussions

By combining the pattern language with an AI planning system, an AI planner which can manipulate the office forms has been developed. Comparing to the previous developed goal-oriented office work representation system, such as POLYMER, the activity schema in this system becomes


Journal-Editing

Title: Author: Received-date:

reviwer	mark

Decision:

Figure 6.

more expressive. For example, the following is an example of the office activity schema: [Croft & Lefkowitz, 1988]

```
ACTIVITY: Accept-or-Reject.Way1
Goal: refereed(?paper)
 Preconditions: member (?paper, papers)
Decomposition: GOAL decision-reached =
                   exists (status (?paper))
       Control: repeat decision-reached until
                   or(status (?paper, "accepted"),
                      status (?paper, "rejected"))
 Agents: ?editor = member (?editor, editors)
```

This activity schema aims to reflect the potential cyclic nature of journal editing. The activity attempts to achieve the goal of reaching a decision on the paper and indicating that this decision is either "accepted" or "rejected". If the decision is anything else, the task will continue. The activity schema has all the information. What is weak here is the link between the activity and the data. More specifically, the use of predicates. Since the predicates can be freely chosen, the link between the activity and the data is loose, and open. And the modelling of the structure of the office data runs a bigger risk of uncertainty. By using the pattern language described in this paper, many predicates can be designed as attributes of office forms, only is a small set of predicates needed. This makes the system more complete, and stable. For instance, in the above example an office form can be designed for the journal editing process, shown as figure 6.

Having defined this form, we can reconstruct the "accept-or-reject" activity schema as follow:

```
<accept-or -reject>
{
office-form: (?paper &(instance-of journal-editing))
goal: (exist (?paper &(decision)))
decomposition: goal decision-reached=
                              (exist (?paper &(decision)))
control: repeat decision-reached until
                    (or (?paper &(decision "accepted"))
                        (?paper &(decision "reject")))
}
```

This activity schema looks similar to the previous one, but its expressiveness is largely strengthened by the direct connection between the activity schema and the office form.

The coexistence of both representations, the office form and the predicates in the system, makes it possible for an office model to model the more routine work by office forms, and the more changeable situation by predicates. This is closer to the practical situation.

References

Bracchi, G., and Pernic, .B., "SOS: A Conceptual Model for Office Information Systems," Data Base, vol. 15, Winter 1984.

Croft, B. and Lefkowitz, L., "A Goal-Based Representation of Office Work" IFIP Conference on Office Knowledge, 1988.

Hayes, P. J. 1975. A representation for robot plans. In the advance papers of IJCAI-75. Tbilisi, USSR.

Liu, H., "Task Oriented Office Form System", Research report for transfer from Mphil to Ph.D, School of Computing and Management Sciences. Sheffield City Polytechnic, England, 1990.

Lum, V., Choy, D., and Shu, N., "OPAS: An Office Procedure Automation System," IBM System J, vol. 21, no. 3, p. 327, 1982.

Tate, A., "Generating Project Networks" In proc. of IJCAI-77. Boston, Ma., USA. [NONLIN]

Tate, A ., "Goal Structure: Capturing the Intent of Plans" Proc. of ECAI-84 Pisa, Italy, 9, 1984 [NONLIN]

Tsichritzis, D., "Form Management," CACM, vol. 25, July, 1982.

Wilkins, D. 1984. Domain independent planning: representation and plan generation. Artificial Intelligence, No. 22. [SIPE]