A RULE-BASED OBJECT/TASK MODELLING APPROACH

Qiming Chen

National Land Information System
Research Institute of Surveying and Mapping, Beijing, China

ABSTRACT

A rule-based object/task modelling approach is proposed which is characterized by specifying object behaviors and domain rules in terms of object-oriented logic programming, and specifying tasks and meta-rules in terms of network- oriented formalism In addition the concepts of associations, virtual objects, multiple level integrity control and net expressions are introduced The object-oriented logic programming system is extended for supporting the semantic modelling, and an explicit control knowledge representation mechanism is developed This approach may be viewed as a step to the integration of object-oriented programming, logic programming, semantic modelling and event modelling, and to the combination of forward chaining and backward chaining techniques Therefore it can provide complementary benifits in deductive query support, integrity control, explicit control knowledge representation and intelligent user interface, and enhance the flexibility and extendability of knowledge based systems to accommodate applications in multiple domains, towards a generalized, rule-based management of data, action and operational schemes This approach is being designed and partially implemented on top of System G [Chen 85b] on a VAX computer

1 INTRODUCTION

The need for developing the next generation information systems has given rise to studies on integrating database systems and Artificial Intelligence (AI) systems [Ullman 85] The fact that both Logic Programming (LP) and Relational Database (RDB) are subsets of first order predicate calculus suggests a resemblence between them On one hand, LP provides powerful formalism for representing database rules and deductive retrievals, on the other hand, RDB provide adequate means of storing and managing knowledge [Gall 81] [Gall 84] [Jarke 84] [Parker 84] [Fuchi 85] [Zani 85]

The message/object model [Cox 84] for object-oriented programming has proven very useful for handling related facts and rules, and maintaining the consistency of the modelled enterprises The object-oriented semantic data modelling approaches provide the benifit of representing the system semantics both structurally and behaviourally [Hammer 78] [Tsic 82] [Wilson 83] [Shep 84] Efforts in developing object-oriented LP system [Zani 84], have also been made

The combination of the above two issues becomes attractive for object/task modelling, because potentially it may combine the benifits of knowledge representation, inference, integrated semantic data modelling and intelligent user interface However, for such a purpose there are still some major problems to be solved

One problem consists in the lack of a rule based and well supported object modelling mechanism that flexibly guides the use of a large knowledge base One way to solve this problem is to extend the relational model [Zani 85] or DBMS [Stone 84], while another way (discussed in this paper) is to develop an LP based front-end for RDB's to support object/task modelling, which consists in making a compromise between semantic modelling, declarative knowledge representation and object oriented LP As tool for that, the present object-

oriented LP should be extended

Another problem with rule-based techniques is the difficulty to express control structures, or meta rules describing the usage of domain knowledge and the linkage of action modules for constructing an operational scheme with desired run-time control As we know, in most of the knowledge based systems currently in operation, control knowledge is not represented explicitly but scattered throughout the procedures, therefore a little bit changing may require modifications to a number of modules Thus these systems are self-contained, support only a single model or a class of models as their configurations are both static and intolerant from a model perspective In order to attack this problem, research on means for handling operational schemes in a data/knowledge base environment are currently active, from the viewpoints of formal foundations [Dahl 82][Gal 84][Chen 85a], integrating action capabilities into RDB systems in terms of the combination of certain loosely coupled techniques sush as triggers, virtual relations and so on, as we described in [Mel 83a-c], behavior semantics modelling [Hammer 78], event modelling [Ant 81], etc Although a number of ad-hoc tricks have been developed, which did bring some improvements from different points, there still lack satisfactory formal solution and integrated system approach to this problem

The focus of this work is to give solutions to the above problems by developing a precise yet flexible rule-based object/task modelling approach

The proposed solution to the first problem consists in developing an integrated object modelling paradigm, which includes the following

Extending the object-oriented LP paradigm [Zani 84] by introducing class-oriented methods (on multiple instances) and multi-class-oriented methods for better supporting semantic modelling and database applications

Introducing the concept of Virtual Objects and its handling mechanism

Developing a multiple level object/task modelling mechanism, where objects, associations and tasks are specified both statically and dynamically with structure, constituents, constraints and methods, in a rule-based fashion

The proposed solution to the second problem consists in modelling the control structure of a problem-solving or decision-making task as a net, and then exploring a rule-based, linguistic and precise net specification formalism and a multiple-level, task-oriented constraints handling mechanism

Under this object/task modelling approach, database applications are specified as localized domain knowledge representation and explicit control knowledge representation Differnt roles an object plays in different tasks can be identified distinctly It offers a unified view to rules, triggers and constraints, handles dynamic operational schemes in terms of general-purpose tools within an integrated system, therefore enhancing the flexibility of knowledge based systems to accommodate applications in multiple domains

This approach is being designed and partially implemented on top of System Q, an extended experimental RDB system [Chen 85b] on a VAX computer

In this paper, starting from overviewing the impacts of logic to RDB through an example, we point out the need to merge the management of facts (data) and the management of domain rules in terms of an object-oriented modelling approach (Section 2), In order to provide a suitable tool, we extend the notion of object- oriented logic programming to better support semantic representation and database applications, and then propose a rule-based object modelling mechanism, in addition with introducing the concepts of associations, virual objects and task (Section 3), To specify the control structure of a task, we combine the techniques of forward chaining and backward chaining, to develop a net-oriented task modelling formalism, thus the rules vs objects and the meta-rules vs tasks are both covered (Section 4), After demonstrating this approach with a task specification example, finally in Section 5 we give some concluding remarks

2 A RULE-BASED FRONT-END TO RDB

RDB and LP techniques are moving fast towards a common destination This holds true both in terms of functionality and performance, and is made possible both from their common ancestry of mathematic logic and the complementary benefits they can provide The studies on the mathematic foundation of RDB and LP have issued a new research field building logic front-ends to database systems for semantic rules representation and deductive retrievals

Logic, as a formal system, relies upon an object language based on the first order predicate, a semantics or interpretation of formulas in that language and a proof theory [Shoe 67] Well formed formulas (wff's) are defined either as atomic formulas or by connecting or quatifying other wffs The general form of clauses that will represent database facts and deductive rules is

P1 & P2 & & Pk -> R1 ! R2 ! ! Rq

Logic admits a declarative semantics, that is, logic clauses may be interpreted as statements of facts In addition, logic admits an imperative semantics in which logic clauses may be interpreted as commands An attempt to answer a query is referred to an attempt to satisfy a goal (or to prove a goal)

As we traditionally identify association between domains and represent them using relations, we will now specify the association between relations and represent behavioral properties by using logic clauses The correspondence between logical predicates and relations is that a predicate could reasonably be stored as a relation, and a hierarchically constructed predicate can be implemented as relations with a type hierarchy (such as an aggregation) We asseme that all variables in a clause are universal quantified (readers may realize that the LP language shown below in the example is a Prolog-like one, where lower case letters stand for constants and predicate symbols, upper case letters stand for variables)

We shall illustrate the concepts described above through an decision-making example concerning a simplified manufacturing order processing system for an instrument assembling company The information needed is stored in certain relations (predicates), includes "order" (states orders), "asmb-des" (lists the parts needed for each model of machines, as well as the assembly time), "parts-inv" and "products-inv" (describe the inventories of parts and ready-made machines), and "parts" and "machines" (descride models and costs), whose instances are viewed as facts of the following form

```
order (order_id, model, qty, finish_qty)
parts (code, cost)
machines (model, cost)
parts_inv (code, name, amt)
products_inv (model, name, amt)
asmb_des (model, asmb_hrs,
     config(senser, cpu, monitor, frame))
```

The rules for processing a single order are stated in terms of Horn clauses

```
order_proc (ORDER_ID  MODEL, PICK_QTY, 0  0  SALES_VALUE)  -
   order (ORDER_ID, MODEL, QTY, FINISH_QTY),
   products_inv (MODEL, _, AMT),
   machines (MODEL, COST),
   AMT >= (QTY - FINISH_QTY),
   PICK_QTY is QTY - FINISH_QTY,
   SALES_VALUE is PICK_QTY * COST, !

order_proc (ORDER_ID  MODEL, PICK_QTY,
            ASMB_QTY, WAIT_QTY, SALES_VALUE) -
   order (ORDER_ID, MODEL, QTY, FINISH_QTY)
   products_inv (MODEL, _, AMT),
   asmb_des (MODEL, _, config(CODE_S,CODE_C,CODE_M,CODE_F)),
   parts_inv (CODE_S, _, AMT_S),
   parts_inv (CODE_C, _, AMT_C),
   parts_inv (CODE_M, _, AMT_M),
   parts_inv (CODE_F, _, AMT_F),
   machines (MODEL, COST),
   PICK_QTY is AMT,
   N is QTY - FINISH_QTY - AMT,
   min (ASMB_QTY, [AMT_S, AMT_C, AMT_M  AMT_F  N]),
   WAIT_QTY is N - ASMB_QTY,
   SALES_VALUE is (PICK_QTY + ASMB_QTY) * COST

wk_ticket (MODEL, ORDER_ID  ASMB_QTY  WK_HRS)  -
   order_proc (ORDER_ID, _  _  0  _, _)  !, fail

wk_ticket (MODEL, ORDER_ID  ASMB_QTY, WK_HRS)  -
   order_proc (ORDER_ID, MODEL  _  ASMB_QTY, _, _),
   asmb_des (MODEL, ASMB_HRS, config(_, _, _  _))
   WK_HRS is ASMB_QTY * ASMB_HRS

parts_req (ORDER_ID  req(CODE, QTY))  -
   order_proc (ORDER_ID, _  _, _  0, _)  !  fail

parts_req (ORDER_ID  req(CODE, QTY))  -
   order_proc (ORDER_ID, MODEL, _, ASMB_QTY, WAIT_QTY, _)
   asmb_des (MODEL, _, config(CODE_S CODE_C,CODE_M,CODE_F))  !,
   parts_inv (CODE  _, AMT),
   (CODE = CODE_S, CODE = CODE_C  CODE = CODE_M, CODE = CODE_F),
   QTY is ASMB_QTY + WAIT_QTY - AMT,
   write (parts_req (ORDER_ID, req(CODE, QTY))), fail
```

where predicate "order-proc" represents the decisions for processing the order, it describes the number of machines which can be delivered immediately (PICKQTY), those that need to be assembled (ASMBQTY), and those that have to wait due to the lack of parts (WAITQTY), and the current SALESVALUE from this order, "work-ticket" lists the tasks of the assembly workshop, "parts-req" lists the net quantity of each part that must be supplied to complete the order

Then one may make deductive queries in a style not possible in traditional RDB systems, such as

```
?- order_proc (0503, _, PICK_QTY,
                ASMB_QTY, WAIT_QTY, _)
```

(For order 0503, how many machines can be directly picked? how many can be assembled right now ? how many have to wait due to the lack of parts?)

```
?- parts_req (0503, req(CODE, QTY))
```

(How many parts must be supplied for processing order 0503 ?)

```
?- wkticket (_, 0503, _, HRS)
```

(Based on the current inventory, how long the assembly work takes ?)

From this example we can see that logic clauses can be employed as a very elegant formalism for representing deductive queries, where the details of which relation are actually stored is hidden from users by the "logic nevigation", and rules can be thought of as a generalization of the derived view mechanism which provide for their efficient implementation via the usual mechanisms for query support

However, the difficulty to express control structures is one of the problem with general first-order logic as a knowledge representation formalism Thus we need to explore in addition an approach to the integrated management of facts and rules, which efficiently and correctly guides the use of a large knowledge base In other words, we need combine the power of logic front-end of RDB with the capabilities of semantic modelling and task controlling, to form an integrated object/task modelling

mechanism which offers a complete view to objects, localizes rules to the domains or tasks they apply, and represents control schemes explicitly

3 RULE-BASED OBJECT MODELLING

3 1 AN EXTENDED OBJECT-ORIENTED LOGIC PROGRAMMING SYSTEM

In the object-oriented semantic modelling paradigm, reality is represented in terms of objects as well as their relationships Each object has an associated set of procedures called methods denoting its dynamic behaviors The manipulation of objects is made by applying the methods on them, referred to as "sending the objects a message"

Unlike the usual operator/operand model, which treats operators and operands as if they where independent, in the above message/object approach, an object instance records its type (class) explicitly, which is used to determine the set of legal operations on the objects of this class, and the type dependencies become permanently encapsulated within classes This concept is coincident with a "save" database design principle localization [Rid 83]

In order to build an object-oriented paradigm as the top of the proposed system, we must first build such a paradigm on the logic front-end layer of the system Generally we identify an object structurally as

a primitive one which can be fully defined without reference to other objects, and represented by an LP predicate without any nested predicate as its argument

a compound one composed in a nested or recursive fashion from two or more objects, represented by an LP predicate with at least one predicate as its argument

Then each method is stated as an LP clause we classify three kinds of methods the individual-oriented methods (i-methods), the class-oriented methods (c-methods) and the multi-class-oriented methods (mc-methods, described in Section 3 3)

An i-method is defined on a single object instance (universally quantified), although that instance may be a complex structure (such as an aggregation) based on a type hierarchy Passing a message is specified by using the infix operator " ", as (the number of arguments may be zero)

```
i-method   object (arg1, arg2,   )
```

A c-method is defined on a class of object instances (the simplest example is a relation with multiple tuples) Its application is represented as

```
c-method   object ()
```

The introduction of multiple instances oriented methods is significant to data-base applications where data extensions are usually much bigger then intensions

In the object-oriented semantic modelling paradigm, an important feature is the inheritance network whereby an object can be declared as a specialization of other objects, therefore inheriting their behavioral properties This feature allows new classes to be built on top of older, less specialized classes instead of being rewritten from scratch The inheritance network is declared by means of the infix operator "isa" The clause

```
object_A  isa  object_AA & object_BB &
```

implies inheritance from A to its ancestor AA or BB all the methods and constraints The "isa" relationship is not symmetric, but transitive, in a sense that an "isa" link can be resulted from the transitive composition of others without redundant declarations

The interpretation of a message consists in the unification of the objects, the unification of the method and the proof of the method If an object is declared by "isa" as an subobject, then the unification of the method with the methods associated with the ancestors of the object in the "isa" network, is necessary When an object has more then one ancestors, we first determine the order of the ancestors, then use a so called "upward-first" search strategy, starting from checking the first available ancestor, observe to see whether it is a root (no further ancestor remained), if not, move upward until either the unification succeds or backtrack to try the next possible ancestor at the highest possible ancestry level

3 2 OBJECT-SPECIFICATION

An object type is the structural specification of a class of objects, which may be hierarchically decomposed The specification of an object type describes its name, structure, methods and constraints

The structure defines the attributes and constituents possessed by that object, Methods consist of the rules applicable to the objects in term of Horn clauses, Integrity constraints are assertions that instances of objects are compelled to abey, which can be classified according to various criteria, sach as type constraints, dependency constraints and so on In our approach a constraint is specified similar to the body of a Horn clause Thus for example the object type "order" is specified as

```
object_type
 order (ORDER_ID, MODEL, QTY, FINISH_QTY)
 {
  structure
    { ORDER_ID      c4,
      MODEL         c8,
      QTY           i4,
      FINISH_QTY    i4
      key   ORDER_ID
    }
  constraints
    { MODEL [ "hc20, hc40, hc60",
      QTY > 0, QTY < 10000,
      QTY > FINISH_QTY
    }
  i-methods
    { what_kind_order ([instrument,MODEL])  -
        order (_, MODEL, _, _)
      unfinish_qty (X)  -
        order (_, _, QTY, FINISH_QTY),
        X is QTY - FINISH_QTY
    }
  c-methods
    { diff (IDa, IDb, X)  -
        order (IDa, _, QTYa, _),
        order (IDb, _, QTYb, _),
        X is QTYa - QTYb
    }
 }
```

The specified constraint means that

(∀ORDER_ID)(∀MODEL)(∀QTY)(∀FINISH_QTY)
(order(ORDER_ID,MODEL,QTY,FINISH_QTY) -->
MODEL ∈ {hc20, hc40, hc60} & QTY > 0 &
QTY < 10000 & QTY > FINISH_QTY)

An instance of an object is a predicate with all the arguments been instantiated with specific values Thus the goal

```
?- unfinishqty(X)   order(0056,hc40,30,10)
```

succeeds with X = 20

And if we have facts

```
order (0001,hc40,40,10)
order (0002,hc40,10, 0)
```

Then the query

```
?- diff(0001,0002,X)   order(0001,hc40,40,10)
```

has no defined meaning, and the goal

```
?- diff (0001,0002,X)   order ()
```

succeeds with X = 30

The inheritance complexity provides links which connect types to generic types, allows methods and constraints on a generic object type to be applied to the instances of all types of objects under a same ancestor For example

```
object-type  inventory (CODE, NAME, AMT)
 {
  structure
    { CODE    c8,
      NAME    c12,
      AMT     i4
      key     ITEM
    }
  constraints
    { AMT >= 0, AMT < 5000
    }
  c-methods
    { clean  -
        retractall (inventory (_, _, 0))
    }
 }
    /* drop records of out-of-stock items */
```

Then we can specify that

```
  products_inv (MODEL, NAME, AMT) isa
  inventory (MODEL, NAME, AMT)

  parts_inv (CODE, NAME, AMT) isa
  inventory (CODE, NAME, AMT)
```

Thus ?- clean parts_inv

drops from the database all the instances for out-of-stock parts

3 3 OBJECTS ASSOCIATION

Although in general everything could be viewed as an object [Wilson 83], It is best to consider only the semantically meaningful objects In fact, the need to represent associative behaviors of a group of different type objects, without creating any "fuzzy" object, does arise in many situations For this purpose we introduce a concept called "association"

Briefly speaking, an association is a group of related objects of different types An association type is the structural specification of a class of associations, and specified by its name, constituents objects, methods and constraints

The constraints, which involve multiple objects, are specified against one (or more if necessary) selected object by means of the infix operator "withcons" The methods of an association are referred to as multi-class-oriented methods (mc-methods), which are defined on multiple classes involved Thus for example an association productsinvvalue is specified as

```
association_type  products_inv_value
 {
  objects
    { machines, products_inv
    }
  mc-methods
    { products_inv_value (MODEL, VALUE)  -
      machines (MODEL, COST),
      products_inv (MODEL, NAME, AMT),
      VALUE is COST * AMT
    }
```

```
constraints
  {  products_inv  with_cons
       {products_inv (MODEL, NAME, AMT),
        machines (MODEL, COST),
        COST * AMT < 500,000
       }
  }
}
```

The specified constraint means that

```
(∀MODEL)(∀NAME)(∀AMT)(∀COST)
(products_inv(MODEL, NAME, AMT) &
machines(MODEL, COST) -->
(COST * AMT < 500,000))
```

The above method shows how to calculate the inventory value for a kind of machines, for example, suppose we have the facts

```
products_inv (hc40, thermal_meter, 30)
machines (hc40, 250)
```

the goal

```
?- products_inv_value (hc40, X)
   products_inv (), machines ()
```

succeeds with X = 7500

The constraint stated above restricts the inventory value of any kind of machines, with a proper alarm facility, this information can be utilized for the manufacturing planning

It is worth noting the difference between a compound object and an association The former, represents a meaningful object and ties to a certain data structure, the later, on the other hand, neither introduces any new object, nor reffers to any storage structure, it only mentions the dynamic behaviors caused by a group of different types objects Thus the introduction of association can improve the system expressive power without having to dream up tedious extra objects

3 4 VIRTUAL OBJECTS

Passing message implies that certain action may be taken ruled by the specified method clause, then the resulting predicate (on the left side of the clause) can be reasonably viewed as a virtual object (VO) The instance of a VO is derived from the instances of other (source) objects, and instantiated by passing message thus represents the results of the message passing Thus VO's may be viewed as a link between data access and data processing Differnt from actual objects, VO's may or may not be actually filed and physically stored until needed

For instance, in the above order processing example, the predicate orderproc can be specified as a VO instantiated by applying the method (identified by the same name) on an association of objects

```
order_proc (ORDER_ID, MODEL, PICK_QTY,
      ASMB_QTY, WAIT_QTY, SALES_VALUE)
order (), products_inv (), asmb_des (),
      parts_inv (), machines ()
```

The declaration of a VO contains structure (which can be the same as for the usual actual object), source objects, mapping rules and so on The source objects can be actual or VO or a mixture of both, thus a VO may be defined hierarchically We list below a VO specification "partsreq", in the order processing example, where one of its source object "orderproc" itself is a VO

```
Virtual_object_type  parts_req (ORDER_ID  req(CODE, QTY))
{
 structure
   { ORDER_ID   c4,
     req (CODE  QTY)
         { CODE   c4  QTY    i4  key    CODE }
     key   ORDER_ID
   }
 source
   { asmb_des  order_proc
   }
 mapping rules
   { parts_req (ORDER_ID, req(CODE, QTY))  -
     order_proc (ORDER_ID, _, _, _, 0, _), !  fail

     parts_req (ORDER_ID  req(CODE, QTY))  -
     order_proc (ORDER_ID, MODEL  _, ASMB_QTY  WAIT_QTY  _),
     asmb_des (MODEL, _, config(CODE_S CODE_C CODE_M, CODE_F)),
     !
     parts_inv (CODE, _, AMT),
     (CODE=CODE_S  CODE=CODE_C, CODE=CODE_M, CODE=CODE_F),
     QTY is ASMB_QTY + WAIT_QTY - AMT
     write (parts_req (ORDER_ID  req(CODE  QTY)))  fail
   }
```

Unification is the way to make VO's physically accessible In the case that a VO has virtual source objects, the latter must be instantiated in advance accor- ding to their own definitions, and such a process may spread upward in the virtual object network until all the actual and VO's involved are fully instantiated Syntactically, the instantiation of a VO, such as "partsreq", is represented by the clause

```
        mapping (parts_req)
```

It also displays and stores all the instances (tuples) of the VO

The instance of a VO has a life time which extends over a single task Its refreshment may adopt one of the following strategies (1) when it is accessed after the alteration of its source objects, (2) by demand During a tast, the cost of recomputation VO's can be reduced by introducing a mechanism called "life flag" The life flag is a property of a VO, indicating whether this object has a valid (live) current instance For each actual or virtual object a possible list of its "directly dependent virtual objects" (DDVO) is generated automatically by the system from all the VO declarations For an actual object, any updating automatically triggers a kill operation, turning off the life flag of its direct dependents (if any) For a VO, any kill operation propagates through its own direct dependents (if any) Thus updating an

actual object will kill all its virtual dependents hierarchically Thereby the recomputation of a live virtual object, which may be directly accessed or indirectly referenced as the source object of others, can be eliminated

It is worth noting that

Both methods specification and constraints specification can be added to the VO specification when necessary

The constraints defined on a VO is interpreted as the post-condition of its mapping (for validating the instance)

No update operations are defined on VO's

This approach offers the following advantages

(1) In the rule-based environment, a VO mapping is just one or a cluster of rules However, a VO is a more feasible entity than a rule a VO has a structure definition, can be directly queried at a high level, with a printing format, It can, flexibly, either be used as a source object for other VO's, or be involved in associations and tasks

(2) VO provides a natural way to accomplish rule localization

(3) Duplicate unification for VO's in a task can be reduced

3 5 MULTIPLE VIEWS TO AN OBJECT

For a knowledge based system, it is reliable and convenient to localize rules on the proper problem domains We shall call this concept "localization", which may cause multiple views to an object That means the roles an object plays may vary from task to task Its behavior is based on its general properties, but may or may not be all the same in different tasks For example, a certain method on an object is significant in one task, but meaningless in another, a constraint on an object is stronger in one task than in another, and so on For instance, a semantic constraint on "order" might be "QTY > 0", where a task-oriented constraint might be a weaker one "QTY > 10" expressing a company policy requiring all the orders must be in batch of minimum 10 pieces

We will see below in the task specifications that the task-oriented behavior of an object is determined by its task-oriented methods and constraints, which can be accessed to show the different roles the object plays in different tasks

In general, the described structural and behavioral object modelling mechanism fully supports semantic data models in a simpler and more versatile framework

4 RULE-BASED TASK MODELLING

A task is an operational scheme for complex decision-making, problem-solving, simulation or control in a knowledge based environment The task modelling concerns not only the domain knowledge, but also the control structure whose description may be viewed as certain meta-rules In this section we will propose a task modelling formalism for the integrated task specification and execution

4 1 FORWARD AND BACKWARD CHAINING

In general a rule "if X then Y" can be used in computations in different ways On one hand, one can try to prove a hypothesis Y by establishing the preconditions X through backtracking inference On the other hand, one can try to match a given situation to the condition X in order to infer a possible action Y through forward chaining Although inference applies backtracking, control structures are easy to be represented by forward chaining

our solution for handling tasks is characterized by using the combination of both methods The task scheme specification is basically a forward chaining network with a number of paths

Thus for an individual action, when a request cannot be satisfied directly, the system will determine if a rule in the knowledge base can be used to reformulate the request, that is, using backward chaining, from the desired data toward database facts which must be ascertained However, the control of a task involving multiple actions is accomplished using forward chaining or triggering mechanism, to tie the actions together, determine the order in which the anticipating actions take place and form an operational scheme

4 2 TASK SPECIFICATION

A task specification concerns about the domain knowledge as well as the control knowledge for the operational scheme As we know, the behavior of an operatio- nal system in a dynamic environment, concerned with states and state transitions, can be modeled as a digraph or a net (such as a Petri-net) Adopting this concept, we propose a task specification approach where the control flow network is specified as a net, the rules for paths of the net are specified as linkers Below we give an overview to the proposed task specification

```
task_name
  {  object_specification
       { actual_objects
         virtual_objects
         methods
         constraints
       }
     scheme_specification
       { actions
         net
         linkers
       }
  }
```

The object specification comprise the domain knowledge of the task It, on one hand, lists all the anticipating objects (which may be actual or virtual, and previously defined with their structures, methods and constraints), on the other hand, describes the task-oriented methods and constraints, which are encapsulated within the task thus can only ba utilized locally for this task This information describes the knowledge base that contains facts and rules about the enterprise to be modelled, defines the system state, and implies the contents of the possible system recovery

The scheme specification concerns the control knowledge for the task, determining the usage of domain knowledge and the linkage of actions which make up an operational scheme with desired run-time control (for more details, refer to author's another paper [Chen 85b])

(1) Action Specification

In the object-oriented paradigm, actions are denoted by the messages (methods on objects) to be passed, that carry out necessary inference, update the system state by resulting in the addition of new facts, or modifications of the existing ones An action specification gives the above correspondence, with the form

action (ACTION, MESSAGE)

We distiguish the concepts of "action" and "message" as following a message is a static description whereas an action is the execution of a method on an object or a group of objects where it is defined A message passing at different time (with different system states) must be identified separately, except in the case of a loop Thus there is a one-to-many mapping from the space of methods to the space of actions An action may be activated when its turn comes and the pre-conditions are satisfied, whose result may be accepted according to the post-conditions

(2) Linker Specification

Linkers describe the conditions for each possible path in the net denoting the task scheme, that is, the connection rules Each linker has four arguments (attributes), as

linker (LINKER_NAME, LINK_MODE, TERMINATE_MODE, CONDITION)

The LINKMODE may be 'r' (regular) or 'c' (conditional branch) If the mode is 'r', a set of rules may be specified under the attribute CONDITION to determine whether the execution may continue or be terminated If the mode is 'c', then more than one set of rules must be specified to determine a proper branch The TERMINATEMODE includes 'h' (hard termination if errors are detected, with system state recovery) and 's' (soft termination without state recovery) In fact linkers can include the pre-coditions and post-conditions of all the actions involved

(3) Net Specification

A net definition specifies the internal linkage among the actions and the control flow (not data flow, as a rule, all inter-action communication must be carried out by accessing data/knowledge base, which may be viewed as "mailboxes") in terms of high level, linguistic, net expressions

The proposed net specification system, with its interpretation, is characterized by introducing the following concepts

net-structures, denoting actions, linkers and sub-nets which are represented by by expressions, and

net forming operations for constructing new nets from existing ones, which map net-structures to net-structures in general

This approach is founded on the use of a fixed set of combining forms called path forms A new net can be built from existing ones by means of path forms and simple definitions Generally a net specification utilizes the following

- a set of net-structures,
- a set of path forms, viewed as operations whose parameters are net-structures,
- a set of sub-net definitions,
- a naming rule called "first-meet rule" utilized in loop specifications

An net-structure may be

- an atom (terminal symbol) which is either an action, a linker, or a special system symbol such as a stop symbol '*'
- a net-expression (non-terminal symbol) constructed in a nested or recursive fashion from net-structures

A path form is an expression denoting a combination of net-structures which depend on the actions, linkers and sub-nets that form the parameters of the expre- ssion Instead of describing nets structurally, path forms represent nets func- tionally Since in general all the net-expressions

are built in terms of path forms, they provide powerful means of representing various net constructions Examples of a basic set of path forms are given below (the capital symbols generally stand for net-structures which may be actions, sub-nets, or linkers)

Composition

A > B	means that A and B are composed from left to right with no linking condition specified
A : (P)B	means that A and B are conditionally composed through a linker P

Serialization

[A, B]	means that A and B are serializable (can take place in parallel)

Condition

(P){A, B, C}	means a conditional branch (conditions are specified by the linker P in front)

compose-to-all

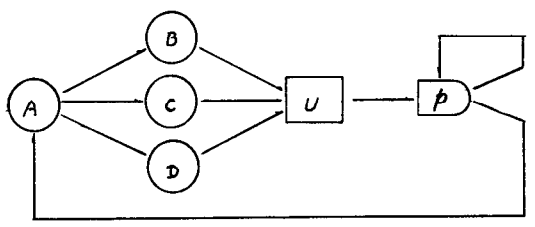
A >: {B, C, D} means [A>B, A>C, A>D]

These path forms can be utilized hierarchically or recursively for representing a complex net as the combination of certain simpler (or lower level) nets and in fact more additional path forms may be defined and implemented

A defined sub-net in a net-expression, and indicated, in the current implementation, by its name preceded with a symbol "@" such as "@A" To apply a defined sub-net means to replace it by its definition in the net expression where it appears

The first-meet rule is a naming mechanism describing how to handle a symbol for an action, a linker or a sub-net, which appears more than once in a net-expression, by going back to the leftmost position in the expression where the symbol appears This rule implies that a back loop is specified by a naming convention, and actions corresponding to the same action module must be named differently except to indicate a loop The "goto" destination of a loop may be an action, a linker or even a sub-net For example, in the net-expression

A : [B, C, D] > @U > (P){A, (P)}



a condition is specified in the first linker (P), indicating two possible subsequent actions go back to the leftmost A, or repeat the checking specified by the leftmost (P), depending on the conditions specified at that (P) and the state of the system at the moment the operation reaches that point Thus the checking action is continuously repeated until some other action changes the system state to let the conditions be satisfied In this example, U is a sub-net-expression and represented graphically by a black box

There are some restrictions which a "well defined" net must satisfy For example, the net-expression [A, C] > (P){A, D} is illegal since in the serialization form [A, C], actions A and C are not serializable when control is transferred from the A on the right to the A on the left

Generally a net specification is a predicate (relation)

net (NET_NAME, NET_EXPRESSION)

where the main-net of a task is identified by the taskname

The author believes that the proposed net specification formalism is considerably simpler than other languages describing a task representable by a 2D digraph, for it lifts the net specification from a structural level to a functional level, and only uses the most elementary naming mechanism, including net definition and substitution rule, as well as the first-meet rule Most importantly, it handles names as objects which can be combined with other objects without special treatment

4 3 TASKS AS A TYPE OF SPECIAL OBJECTS

In this approach, the "task" is treated as a special type of system objects An task is a structure or an abstract type filled with information about its domain and control knowledge (action, net and linker specifications) as its instance value This information is stored as a set of associated relations with fixed intensions which consist of the intension of the "task" type There are standard operations on "task" variables, such as most of the query and update operations on the associated relations, and the global query and execution operations on the whole task If properly specified the execution of a task is automatically carried out by a system called Process Handler (PH) once it is activated This approach is also useful for developing knowledge based "Task Libraries"

4 4 AN EXAMPLE

To illustrate the proposed approach, we will give a complete rule-based task specification for the order processing example mentioned before, as following

```
task order_processing
  {
   object_specification
     { actual_objects   { order, products_inv, parts_inv, asmb_des, parts, machines
                        }
       virtual_objects { order_proc, parts_req, wk_ticket
                        }
       mc-methods   { update_inv (MESSAGE)  -
                       order_proc (ORDER_ID, MODEL, PICK_QTY, ASMB_QTY,
                                   WAIT_QTY, SALES_VALUE)),
                       asmb_des (MODEL, _, config(CODE_S, CODE_C, CODE_M, CODE_F)),
                       retract (products_inv (MODEL, NAME, AMT)),
                       NEW_AMT is AMT - PICK_QTY,
                       assert (products_inv (MODEL, NAME, NEW_AMT),
                       retract (parts_inv (CODE_S, NAME_S, AMT_S)),
                       retract (parts_inv (CODE_C, NAME_C, AMT_C)),
                       retract (parts_inv (CODE_M, NAME_M, AMT_M)),
                       retract (parts_inv (CODE_F, NAME_F, AMT_F)),
                       NEW_AMT_S is AMT_S - ASMB_QTY,
                       NEW_AMT_C is AMT_C - ASMB_QTY,
                       NEW_AMT_M is AMT_M - ASMB_QTY,
                       NEW_AMT_F is AMT_F - ASMB_QTY,
                       assert (parts_inv (CODE_S, NAME_S, NEW_AMT_S)),
                       assert (parts_inv (CODE_C, NAME_C, NEW_AMT_C)),
                       assert (parts_inv (CODE_M, NAME_M, NEW_AMT_M)),
                       assert (parts_inv (CODE_F, NAME_F, NEW_AMT_F)),
                       MESSAGE = invetories_updated

                      update_order (MESSAGE)  -
                       order_proc (ORDER_ID, MODEL, PICK_QTY, ASMB_QTY,
                                   WAIT_QTY, SALES_VALUE))
                       retract (order (ORDER_ID, MODEL, QTY, FINISH_QTY)),
                       NEW_FINISH_QTY is FINISH_QTY + PICK_QTY + ASMB_QTY,
                       assert (order (ORDER_ID, MODEL, QTY, NEW_FINISH_QTY)),
                       MESSAGE = order_updated
                    }
       constraints { order with_cons {MODEL [ "hc40, hc60"}
                     products_inv  with_cons  {products_inv (MODEL, NAME, AMT),
                                               machines (MODEL, COST),
                                               COST * AMT < 500,000}
                    }
     scheme_specification
      { action   { o  is  mapping (order_proc)
                   a  is  mapping (wk_ticket)
                   b  is  mapping (parts_req)
                   u  is  update_inv (MESSAGE)   parts_inv(), products_inv(),
                                                 asmb_des (), order_proc ()
                   c  is  listing (parts_inv)
                   d  is  listing (products_inv)
                   v  is  update_order (MESSAGE)   order (), order_proc ()
                 }
        net      { order_processing  is  (p0) @proc > @update
                   proc              is  o > (p1)[a, b]
                   update            is  u > [c, d] > v
                 }
        linker   { p0  is  (r, h, (order with_cons {QTY > 10}))
                   p1  is  (r, h, (order_proc with_cons {ASMB_QTY > 0},
                                   parts_inv with_cons {QTY > 0}))
                 }
     }
   }
```

We assume that all the anticipating objects are previously declared Within this task, the virtual objects only need to be instantiated once (by "mapping") In fact, the three virtual objects, holding the resulting data of the order processing task, can also be simply queried outside this task, just for estimation, without updating the system This is an evidence showing the flexibility offered by our multiple level object/task modelling approach

The net representation for the task is illustrated in Figure 1 Its performance can be described as follows Action o, a, and b carry out the order processing in terms of computing virtual objects "orderproc", "workticket" and "partsreq", and convey the resulting data to users, actions u and v update the inventories and the order, action c, d display the new

state of inventories after processing the current order Certain control rules are specified at linkers p0 and p1

4 5 MULTIPLE-LEVEL INTEGRITY CONTROL IN AN OBJECT/TASK ENVIRONMENT

An efficient and flexible integrity control facility is quite essential for a generalized system used to support multiple models in an object/task environment The approach we have developed is characterized as follows

(1) An Integrity Control Monitor (ICM) has been developed which is driven by the constraints specifications on objects and tasks The stored facts can only be modified by specified methods, which are guarded and validated by the system These operations form the basic building blocks of application programs thus even a careless user would not have a chance to violate them

(2) In order to have a flexible constraints handling mechanism in a multiple task environment, we classify the constraints by levels to accommodate various requirements Within this flexible framework the user may reconstruct the application schemes with a minimum amount of upheaval We distinguish three level constraints

BC, are the basic constraints specified against objects, which ensure the objects semantically significant They are the strongest type of constraints, global to the whole system and must always hold regardless of individual situations Monitored by the system integrity control facility, it is in principle impossible to carry out an action which violates the BC Any violation of the BC will terminate the action or task (hard terminate)

TC, are the constraints locally specified within a task, and are weaker than BC Thus a system state may meet the conditions on one task, but might not meet the conditions on another In other words,

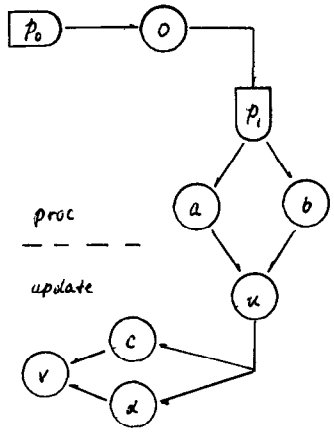


Figure 1
The net representation for the task order_processing

the violation of TC does not imply the violation of BC, thus state recovery is not absolutely necessary, but depends on users' requirements

LC, are specified in the linkers of tasks for triggering certain actions They are weaker than both BC and TC, and are local to the internal linkage of certain taskes

These relationships are detected by the system during the TC and LC specifications, or during an attempt to alter any of them In fact, handling an operational scheme as an integrated object "task" within the system framework makes it more reliable to control the data integration and easier to define task-oriented constraints

5 CONCLUSIONS

In this paper, we have combined logic with the object-oriented semantic modelling, at object, association and task levels We have also complementaryly utilized the techniques of forward chaining and backward chaining, and developed a net-oriented task modelling approach

This work may be viewed as a step to the integration of object-oriented paradigm, logic programming, semantic modelling and event modelling, providing therefore complementary benifits in inference, deductive query support, integrity control, and explicit control knowledge representation, towards a generalized, rule-based management of data, action and operational schemes

There still some problems remained For modelling database applications, Logic clauses do not have the full expressive power of symbolic logic A well known shortcoming is their inability to explicitly represent negative information and if-and-only-if Improving the implementation effeciency is also a issue to be explored

This approach is being designed and partially implemented on top of System G [Chen 85b] on a VAX computer, where the logic front-end is based on "tight coupling" mechanism [Vass 83], and the PH and ICM are modified from their original versions on System G

REFERENCES

[Ant 81] V De Antonellis and B Zonta, "Modelling Events in Data Base Applications Design", Proc of VLDB7, 1981, pp 23-31

[Chen 85a] Q Chen, "Extending the Implementation Scheme of Functional Programming System FP for Supporting the Formal Software Development Methodology", Proc of 8th International Conference on Software Engineering, London, Aug 28-30, 1985

[Chen 85b] Q Chen, "Toward A Generalized Data/Action Management An Approach for Specifying and Implementing Operational Schemes', Proc of 1st Pan Pacific Computer Conference, Melbourne, Australia, Sep 10-13, 1985

[Cox 84] B J Cox, "Message/Object, An Evolutionary Change", IEEE Trans On SE, pp 50-61, Jan 1984

[Fuchi 85] K Fuchi, "The Japanese Fifth Generation Computer Systems Project", Proc of The 1st Pan Pacific Computer Conference, pp 1553-1560, 1985

[Gall 81] H Gallaire, "Impacts of Logic on Data Bases", Proc of VLDB 7, pp 248-259, 1981

[Gall 84] H Gallaire, J Minker and J Nicolas, "Logic and Databases A Deductive Approach", Computing Surveys, Vol 16, No 2, 1984

[Hammer 78] M Hammer and D Mcleod, "The Semantic Data Model A Modelling Mechanism for Data Base Applications", Proc SIGMOD, 1978, pp 26-36

[Jarke 84] M Jarke, J Clifford and Y Vassiliou, "An Optimizing Prolog Front-End to a Relational Query System", Proc of ACM-SIGMOD 84, pp 296-306, 1984

[Melk 83a] M Melkanoff and Q Chen, "An Experimental Database Which Combines Static and Dynamic Capabilities", Proc of Engineering Design Applications, ACM-SIGMOD'83/Database Week, pp 53-61, 1983

[Melk 83b] M Melkanoff and Q Chen, "Integrating Action Capabilities into Information Databases", Proc of 2nd International Conference on Databases (ICOD-2), Cambridge, England, 1983

[Melk 83c] M A Melkanoff and Q Chen, "A Generalized Database Management System for Supporting General Modelling and Simulation", 4th International Symposium on Modelling and Simulation, Lugano, Switzerland, pp 158-162, 1983

[Parker 84] D Parker et al , "Logic Programming and Databases", Proc of 1st Int Workshop on Expert Database Systems, 1984

[Rid 83] D Ridjanovic and J Brodie, "Action and Transaction Skeletons High level Language Constructs for database Transactions", Proc of ACM-SIGPLAN 83, pp 94-99, 1983

[Shep 84] A Shepherd and L Kerschberg, "Prism, A Knowledge Based System for Semantic Integrity Specification and Enforcement in Database Systems", ACM-SIGMOD 84, pp 307-315, 1984

[Shoe 67] J Shoenfield, "Mathematical Logic", Addison-Wesley, 1967

[Stone 84] A Stonebraker, E Anderson, E Hanson and B Rubenstein, "QUEL as a Data Type", ACM SIGMOD 84, pp 208-214, 1984

[Tsic 82] D Tsichritzis, F Lochovsky, "Data Models", Prentice-Hall, 1982

[Ullman 85] J Ullman, "Implementation of Logical Query Language for Databases", Proc of ACM-SIGMOD 85, 1985

[Vass 83] Y Vassiliou, J Clifford and M Jarke, "How does an Expert System Get its Data", Proc of VLDB 9, pp 70-72, 1983

[Wilson 83] G Wilson, E Domeshek, E Drascher and J Dean "The Multipurpose Representation System", Proc of VLDB 9, pp 56-69, 1983

[Zani 84] C Zaniolo, "Object-Oriented Programming in Prolog", Proc of Int Logic Programming Symposium, IEEE 1984

[Zani 85] C Zaniolo, "The Representation and Deductive Retrieval of Complex Objects", Proc of VLDB 11, pp 458-469, 1985