An Extended Metamodel based on Logic & Object-Oriented Approach

박 성주, 오 세창, 이 하빈
(한국과학기술원 경영과학과)

현존하는 대부분의 메타 시스템들은 개체-관계 모형을 기반으로 한 메타모형을갖는다. 그러나 개체-관계 모형을 기반으로 하는 메타모형은 정보 시스템 개발환경에 대한 정보를 충분히 표현하기에 부족하며 유추능력에 있어서도 한계성을 가지고 있다. 본 논문에서는 이러한 단점들을 극복하기 위해 확장된 메타 모형이 제시되었다. 새로운 메타모형에서는 정보를 충분히 반영하기 위하여 객체지향적 데이타 모형을 기반으로 하며 유추능력을 향상시키기 위하여 논리를 도입하였다.

I. Introduction

Software is a major elements in information systems(IS) and the difficulties of IS development by different developers have led to the adoption of software-developement-environment(SDE). The primary purpose of meta system is to generate major parts of particular method based SDE as its instance [DART 91][PERR 91]. The impetus toward meta system approach arises from the facts that method based SDEs are effective only in fairly restricted areas and have extremly similar structures and underling principles.

However, to generate method based SDEs such as specification support system efficiently , contemporary meta systems have some drawbacks. One of their major shortcomings is that they are relatively poor in their expressive power of metamodel that is a core part of meta system. Also, contemporary meta systems should be more intelligent.

Nowadays object-oriented and logic-based Approaches have attracted growing interest among software designers, particularly among those working on knowledge-based applications. Because both offer real advantages over classical data models and traditional programming methods [KOSH 88], we think that object-oriented approach can complement semantic scantiness of current ER based metamodel and logic approaches can take deductive capability easily. In this paper, we combine object-oriented approach with logic approach and design LOOM (Logic and Object Oriented approaches based Metamodel) to enhance the capability of meta model. A prototype meta system, LOOMS, is implemented in Prolog language and C language on PC.

II. Meta System

In general, SDE is defined as a set of software/hardware tools which is provided to developers in software development process. In particular, Method-based environment support particular development methods and management of the development process[DART 87]. Development methods are those used by developers in phases such as requirements analysis, system specification, and design. Contemporary method-based environments have focused to support for requirement specification. For convenience, the term of development environment is used indifferently to that of requirement specification support environment. But, developement environments have some problems with them. i.e. inflexibility and difficulty in integration between systems. And to overcome these limitation some researcher have proposed meta system approach [DEME 82].

As shown in Fig 2 the meta system takes as input a formal specification of an development environment in some formalisms and generates an development environment automatically. There are two levels of meta system use: the meta level and environment level. At the meta level, the development environment is specified with metamodel. And meta system generate development environment. At environment level, the target system is specified with development environment.

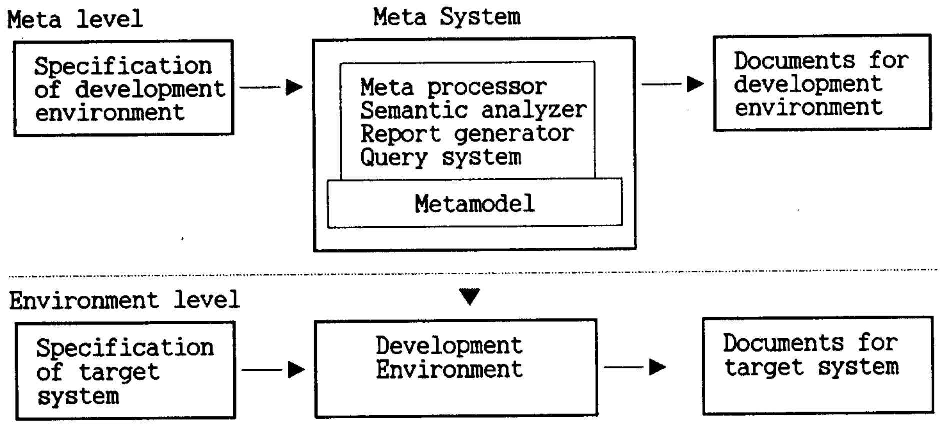

Meta level
Meta System
Specification of development environment
Meta processor
Semantic analyzer
Report generator
Query system
Metamodel
Documents for development environment
Environment level
Specification of target system
Development Environment
Documents for target system

Fig. 1 Conceptual view of a Meta system

Meta system approach has several benefits as follows. 1) Endeavors for development of individual support tool are saved , 2) flexibility for domains is increased, and 3) comparison between individual specification support systems is available.

The Core part of a Meta system is metamodel. Metamodel is meta system's data model which is used to represent, store, and maintain specification for development environment. To capture the information about the static and dynamic components of the S/W developement environments, metamodel must be able to represent following knowledge.

(1) Structural knowledge

- Object
- Relationship

(2) Behavioral Knowledge
(3) Constraint
  - Static constraints
  - Dynamic constraints
  - Method constraints

Most existing meta system have an ER (or extended ER) based metamodel. That is, they mainly concentrate on representing and capturing structural semantics of a method such as static components, their attributes, and interrelationships. This result in loss of knowledge such as behaviors and constraint on structural constructs and behaviors in modeling process. On the other hand, contemporary meta systems should be more intelligent. But it is difficult to make existing E-R based metamodel intelligent . To overcome above drawbacks LOOM is designed.

III. Logic & Object-Oriented Approach

In logic programming, one constructs a program by creating knowledge base of axioms, that is, by saying what is true. The program is executed by entering a theorem and asking the system to find proof given the set of axioms. An application-independent inference engine accepts queries from user, access the facts in its database, and draws appropriate conclusions, in some case, record its conclusion in its database.

Prolog, the most popular example of logic programming style, uses first order predicate logic to derive theorems. Prolog program is a collection of facts and rules. The basic units for building facts or rules are "predication", i.e., expression that say simple things about the individuals in universe of discourse. For example, a piece of information "canaries is a bird" is represented :

```
is_a(canaries,birds).
```

Classes of individuals, which we would introduce in English through words such as "everyone", "anything", and "somebody" must make use of variables. For instances, "Every one is liked by his mother" can be represented:

```
likes(mother(X),X).
```

On the other hand, Prolog rules are if-then statement of the form

```
P1 :- P2,P3,...Pn.
```

For instance,

```
super_Class( X, Y ) :-
            is_a( Y,Z ),
            super_Class( X, Z )
```

means that "IF Y is a Z and X is super class of Z THEN X is super class of Y". Prolog's declarative style provides a natural way to represent rule-based knowledge.

Object-Oriented approach originated in Simula 67 and Smalltlk-80. The heart of the object-oriented approach is the idea of organizing knowledge as a collection of objects. Since interaction between objects are allowed only through a uniform communication mechanism called message passing implementation of each object becomes independent of the implementation of others. This facilitates the modularization of the knowledge of interest. Another important idea is class inheritance. This enables the creation of objects by specializing existing similar objects. This feature contributes to the factoring of knowledge as a class hierarchy. But, it is difficult to represent knowledge of a declarative nature or to develop a method incrementally in these languages.

From the above observations we think that it is one meaningful way to make more powerful meta systems in representation and deduction to adopt these concepts in metamodel. Several efforts have been made to design and integrate a object-oriented model with logic. For example, see [BANC 86][[FUKU 86][Han 85]. In this paper, we have adopted the way to implement object-oriented model in an existing logic environment, because this approach is relatively easy to implement and not require a new language or facilities. Object-oriented concepts which are necessary for metamodel are implemented in Prolog.

IV. An Extended Meta Model

The LOOM's meta type consist of object type, relationship type, and constraint type. Fig 3 shows the relationships of LOOM modeling constructs

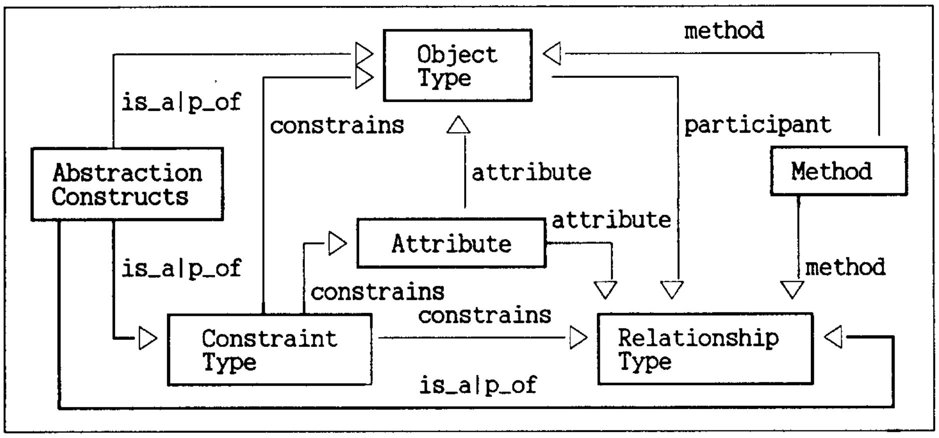


Fig 2. Modeling Concepts of LOOM

1. Object Type

All entities and concepts are represented as objects. Every object belongs to a type, and the objects belonging to the same type have common attributes, and methods. The specification of an object type includes the definition of abstraction, attributes, and method implementation.

```
OBJ_TY        <object type name>  ;
IS_A or P_OF <abstraction definition> ;
ATTR          <attribute definition> ;
```

```
METHOD       <method definition> ;
```

2. Relationship Type

In LOOM, a relationship is represented by relationship type. The relationship type is logical entity which provides an abstract mechanism for the association as a conceptual construct, and capture the semantic of the relationship more clearly.

The relationship type of LOOM includes the information about participating classes, role of participant, and attributes of the relationship itself. For binary relationship, property of relationship type such as symmetry/asymmetry, reflexivity/irreflexivity, and transitivity /intransitivity can be represented as its attributes. Following is the syntax of an relationship definition. It is same to that of object except for participant definition.

```
REL_TY       <relationship type name> ;
IS_A or P_OF <abstraction definition> ;
PART         <participant definition> ;
ATTR         <attribute definition>   ;
METH         <method definition>      ;
```

3. Constraint Type

The constraints are explicitly represented and maintained as constraint type. Semantics for relationships such as dependency ,commonness, and cooperation can be considered as its constraints. For example, the existence of relationship depends on the existence of each participant object types. This existence dependency is represented by constraint.

The syntax of an constraint type definition is :

```
CON_TY <constraint name> ;
ABST   <abstraction definition> ;
CONS   <constraint content definition> ;
```

4. An Example : Data Flow Diagram

In this section, for help the understanding of LOOM, we specify model of data flow diagrams (DFD) which is popular method for specification.

DFD method can be specified by using LOOM. Object types of DFD are 'interface', 'process', 'dataflow', and 'datastore'. Relationship types of DFD are 'in', 'retrieval', 'store', 'transfer', and 'out'. And some constraints are exists.

The first example shows that object type 'dataflow' has an attribute 'description' and its domain is string. At the next example, relation type 'in' is a relation and it's participants, participants' role, cardinality are declared.

```
OBJ_TY 'dataflow' ;
  ATTR 'dataflow' 'description' DOMA {string(X)} ;
```

```
REL_TY 'in' ;
  IS_A 'in' 'relation' ;
  PART 'in' 'interface' ROLE 'sender' CARD 'totalfunctional' ;
  PART 'in' 'dataflow' ROLE 'data_content' CARD 'totalfunctional';
  PART 'in' 'process' ROLE 'receiver' CARD 'totalfunctional';
```

DFD method constraints are explicitly described in LOOM. For example, DFD method should satisfy the followings.
"All relationship type have at least two different participant object types".
This constraint is described as

```
CON_TY 'at_least_2' ;
CONS 'at_least_2' {IF part(X,Y,_,_),part(X,Z,_,_)} THEN {Y \==Z};.
```

V. A New Meta System(LOOMS)

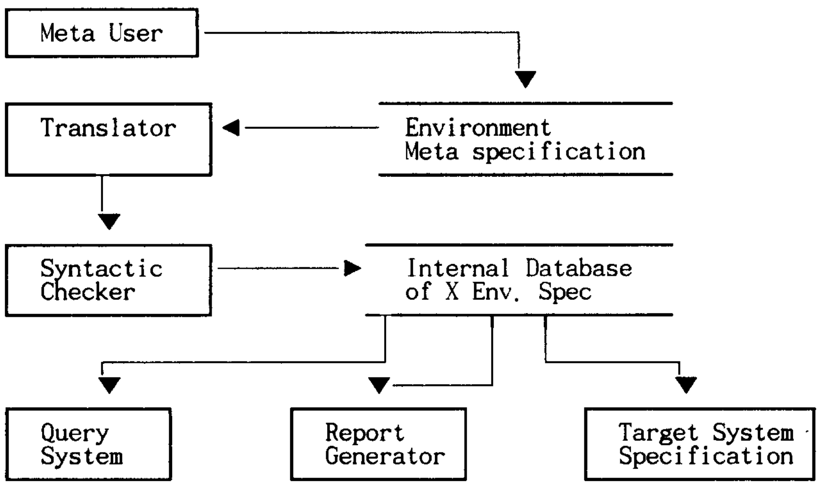
For prototype of meta system with LOOM, Logic and Object Oriented model based Meta System(LOOMS) is designed and implemented in Prolog language and C language under PC environment.

1. Overall Architecture of LOOMS

The overall architecture of LOOMS is shown in figure 3. LOOMS consist of following components.

1) Translator / Prolog Syntactic Checker
2) Query System
3) Report Generator
4) Target System Specification

The descriptions of each components will be stated from now.



Meta User
Translator
Environment Meta specification
Syntactic Checker
Internal Database of X Env. Spec
Query System
Report Generator
Target System Specification

Fig 3. Overall Architecture of LOOMS

2. Translator & Prolog Syntax Checker

(1) Translation Rule

In LOOM all modeling constructs are translated into Prolog Horn clauses. The translation of modeling constructs into Prolog Horn clauses has been taken automatically by use of Translator of LOOMS. To do this, it provide possibility that we can use deductive capability maintaining object-oriented concepts. Translation rules are shown in Fig 4.

```
[R 1]   OBJ_TY <object type name> ;
    =>  obj_type(object_type_name).
[R 2]   REL_TY <relationship type name> ;
    =>  rel_type(relationship_type_name).
[R 3]   CON_TY <constraint type name> ;
    =>  con_type(constraint_type_name).
[R 4]   IS_A <type_name> <super type name> ;
    =>  is_a(type_name, super_type_name).
[R 5]   P_OF <type_name> <super class name> ;
    =>  p_of(type_name, super_class_name).
[R 6]   ATTR <type_name> <attribute_name>
              DOMA {<attribute type>};
    =>  attr(type_name, attribute_name).
    =>  doma(type_name, attribute_name, X) :- <attribute type>.
[R 7]   METH <type_name> <method name> ACT {<method action>};
    =>  method(type_name, method_name) :- <method action>.
[R 8]   PART <relationship_type_name> <participant_type_name>
              ROLE <role name> CARD <cardinality> ;
    =>  part( relationship type name,
                 participant name, role name, cardinality).
[R 9]   CONS <constraint type name>
         IF {<constraint condition>} THEN {<constraint action>} ;
    =>  cons(constraint_type_name) :-<condition>,!,<action>.
```

Fig 4. Translation Rule

(2) Translator & Prolog Syntax Checker

The function of translator is processing the meta specification of specific environment in LOOM as input and producing Prolog clauses of it. This module is implemented by the use of PCLEX/PCYACC. The process of this follows. First, meta specification in LOOM is divided into token and its attribute value by lexical analyzer (PCLEX), and passed to over syntax analyzer (PCYACC) for further processing. Secondary, syntax analyzer receive the stream of tokens and its attributes value as input and parses this stream for the purpose of syntax checking. Finally, translator put the stream of token and its attribute value into predefined Prolog clauses according to translation rules. Then, Prolog syntactic checker verify translated meta specification for Prolog grammar. It's process is similar to that of translation. Checker is also implemented by the use of PCLEX/PCYACC.

3. Query System

In LOOMS, query is processed through embedded inference engine. If user admit query as Prolog clauses, then system try to answer by the use of back tracking and unification. For example, user want to ask "what are attribute (include inherited) of object type 'X'", then following Prolog clause is necessary.

```
?- attr('X',ATTR).
```

Then system answer all attribute of object type 'x' through 'is_a' hierarchy.

4. Report Generator

On user's request, the system can produces the following reports which reflect different aspects of meta specification.

- Object Type Related Report
  - Object Type List
  - Object Type - Abstraction List
  - Object Type - Attribute List
  - Object Type - Method List
  - Object Type - Relationship Type List
- Relationship Type Related Report
  - Relationship Type List
  - Relationship Type - Abstraction List
  - Relationship Type - Attribute List
  - Relationship Type - Participant List
- Constraint Type Related Report
  - Constraint Type List
  - Constraint Type - Abstraction List
  - Constraint Type - IF THEN List

5. Target System Specification

For specification of target system, LOOMS use information of specification of environment. That is, If one specify some aspect of target system, then LOOMS verify its validation on the information of specification of method. For example, when one specify " process 'count' ; ", LOOMS check whether 'process' is object type or relationship type or constraint type. To specify target system conveniently, interactive specification editor can be used.

VI. Conclusion

In specification of development environment, it is required to capture information about behavior and constraint as well as structure. But current meta systems that are based ER model or EER model can specify only structural characteristics supported by ER model or EER model. And they should be more intelligent than now.

This paper proposed a new meta model that is based on logic and object-oriented approaches to surmount drawbacks of ER based meta systems. LOOM (Logic and

Object-Oriented approaches based Metamodel) has combine object-oriented model with logic to utilize both benefits. In our approach, specification of development environment are described by the use of LOOM. Then specification is translated into Prolog clauses. And this specification Prolog clauses are analyzed.

The contributions of this paper are as follows.

First, for behavior and constraint, LOOM is more expressive than ER based metamodel.

Second, checking their syntax grammar, specifications can be automatically translated into Prolog clauses.

Third, by the use of logic, deductive capability is extended on analysis/query about specification of development environment.

We expect that the concept of enhanced meta system should contribute to develope domain dependent specifications and manage them into a unified view.