A rulebase Model for Data and Knowledge Integration in Distributed Heterogeneous Environments

By

Dr. M'hamed Bouziane
Digital Equipment Corporation
ZKO2-3N/30
110 Spit Brook Road
Nashua, NH 03062

Dr. Cheng Hsu
Associate Professor of Decision
Sciences and Engineering Systems
Rensselaer Polytechnic Institute
Troy, NY 12180-3590

December 1991

Keywords: rulebase model, data and knowledge representation, information management, computerized manufacturing.

Abstract

Computerized Manufacturing systems invariably entail multiple data and knowledge resources that tend to be implemented on distributed and heterogeneous computing platforms. Three aspects of integration, therefore, must be addressed in any effort of information integration for these systems: (1) data integration (among, e.g., product design databases, shop floor control databases, and production planning databases), (2) knowledge integration (across various knowledge-based systems that share common enterprise facts, operating rules, or decision logic), and (3) data and knowledge integration (so as to, at least, calibrate databases with factbases and data models with knowledge models).

A key issue in this integration problem, especially the third aspect, is the global representation of knowledge in the form of production rules (including data semantic constraints as well as application processing logic). This paper develops a rulebase model to address this need. In particular, the model combines production rule with relation into a structure that allows full factbase management, inclusion of complete logic (both condition and action), and efficient rulebase processing. While contributing to the generic areas of knowledge base systems, in its own right, the rulebase model is developed as a part of the metadatabase approach to manufacturing information integration [Hsu85, Hsu90, Hsu91, Hsu91]. The Two-Stage Entity-Relationship (TSER) model [Hsu85, Hsu87] is employed for the representation method.

1. Introduction

Data and knowledge engineering traditionally is concerned mainly with single systems, whether or not they are distributed across a network of computers. Therefore, the methods and techniques available previously in the field are not poised to effect computerized manufacturing enterprises: They need new, additional results to integrate at an enterprise level the large numbers of data and knowledge resources they tend to employ and deploy through different paradigms, structures, and platforms. Considerable efforts have been devoted to developing new technology in the past decade with major progresses achieved in virtually all areas. Ever-revealing potential of integration, however, continues to press for additional fundamental results.

Among these needs is the integration of data and knowledge throughout a manufacturing enterprise [CIMOSA, Su, GIRDpaper, ...]. This problem, however, has a generic root in what many researchers refer to as knowledge bases. We examine this root below.

Knowledge Modeling for Integration:

The concept of knowledge base management systems (KBMSs) calls for achieving a more advanced and intelligent form of interoperability in which database and knowledge-based systems can interact to execute tasks jointly [Bro86, Coh89, Man90]. There are a number of basic reasons why a fusion of these two areas would benefit both. First, data models and knowledge representation methods are essentially concerned with two complementary aspects of information: statics vs. dynamics. Thus, when made compatible, much of the data semantics not explicitly represented in the database (e.g., information flow and process logic) can be derived from the knowledge-based processing components of the same enterprise, while the latter can tap into the former for persistent

data in a shared environment [Hsu88, Man90]. Second, the management facilities provided by the database approach could be extended to knowledge-based systems, since most lack a reliable and robust knowledge management capability [Jac90, Sto88a]. Third, in a similar way, the database can utilize the knowledge processing facilities to enhance its capacity for handling event-triggers, integrity control, and other built-in procedures. Finally, from a broader perspective, the emerging concept of information integration requires new metadata technology whereby data resources and contextual knowledge are unified for information management [Hsu91a, Hsu91b].

However, there is a fundamental barrier to the realization of such integration: knowledge modeling (as compared to data modeling). Despite the many advances in database and knowledge-based technologies in the past decade, knowledge modeling is still largely an unresolved issue. The importance of conceptual modeling of knowledge in multiple-application environments has only recently been explicitly formulated [Hsu88, Pot88, Su90]. Moreover, building a KBMS involves integrating various information resources, including heterogeneous distributed databases, knowledge-based systems, and conventional application programs [Man90]. Knowledge-based systems themselves may involve heterogeneous knowledge representations, thereby necessitating a common knowledge base schema that consolidates these representations. Consequently, without the ability to model both data and knowledge in a unified manner, data and knowledge integration cannot result and the concept of KBMSs will remain largely a vision rather than a reality [Hsu88].

A case in point is the rule-based approach to knowledge processing, which is a proven technology with flexibility and other widely recognized features [Cac90, Kei90, Sel88, Tou85]. Numerous expert systems in various application domains have been developed using this approach; however, their production rules have rarely been abstracted

or consolidated into models. Not only is there a lack of a framework for recognizing and representing inter-relationships across individual expert systems in an organization, but also there is no structure to control the propagation and maintenance of the consistency of the rules within an expert system [Bry89, Hsu88]. As a result, production rules of one system cannot be utilized or cross-referenced by other systems even when all of the systems in question are interrelated (e.g., part routing and job scheduling in a CIM environment) and should share the same body of knowledge (e.g., workstations' operating rules). Therefore, when applications will inherently call for evolution of the production rules, this lack of integrity control simply represents a major limitation to the rule-based approach, since effective management of changes in the rules will be a central requirement in this case. New results on solving this modeling problem are necessary before the knowledge processing requirements in enterprises such as CIM can be satisfied.

Rulebase Model:

Before examining the issues of modeling, it is helpful to distinguish the levels of abstraction indigenous to data and knowledge. Fundamentally, data represents raw facts in an application, whereas knowledge refers to information about those facts, such as the intended use of the facts and how to use them. Therefore, knowledge is by nature metadata (describing data) and its integration with data should be best realized at the metadata level [Dee86, Hsu87c]. In other words, knowledge should be integrated with other classes of metadata such as data structures and data semantics, as opposed to being united directly with raw data themselves. With this understanding, the integration problem is addressed at a semantic level (or a higher level of abstraction than instances of data classes) congruent to the nature of a unified structure for data and knowledge representation and processing.

In a fundamental way, individual rule-based systems are similar to stand-alone file systems in data processing. The above-mentioned need for rulebase modeling in expert

systems is also comparable to the problem that used to plague file systems before the advent of database technology. The classical data management issues such as data sharing, update integrity, and data independence have been successfully addressed through data modeling and database design methodologies. Thus, a similar philosophy can be and should be employed to address the integration of the different knowledge-based systems in an organization.

Towards this end, a rulebase approach was envisioned [Hsu88] which entails: (1) an architecture similar to the three-schema design of database, (2) a rulebase model and its attendant modeling methodology for rule schema design, and (3) a rulebase management system implementing the rulebase. This approach is incorporated into this research as the basis for developing new results. Furthermore, the rulebase model is also intended to set the stage for developing effective linkages with database methods, as well as with other knowledge representation methods, so as to provide a foundation for the emerging technology of KBMSs.

The new results developed in this research focus on rulebase modeling and processing. The Two-Stage Entity-Relationship (TSER) method [Hsu85] is employed to explore its promises of representing rules as entities and relationships, just like data representations in TSER [Hsu87]. This way, two separate but synergistically related modeling paths are formulated, whereby the Subject-Attribute-Entity-Relationship model representing data and the Subject-Context-Attribute-Rule model representing the rules. The integration of these two models is achieved through common constructs at the structural level of TSER. The rulebase modeling approach itself is presented below in Section 2, with a detailed development of the basic representation method included in Section 3. The overall rulebase model is then discussed in Section 4, illustrated with examples in Section 5, and compared with previous methods in Section 6. Section 7 concludes this paper with

a review of the rulebase model in the context of the generic KBMS as well as that of the metadatabase approach to information integration.

2 The Modeling Approach

The rulebase model is anchored in three basic results established in the literature: (1) the metadatabase model for overall metadata integration, (2) the TSER model for information modeling primitives (metadata classes), and (3) the expression grammar, along with predicate logic, for production rule structures. The basic approach used in modeling knowledge in the metadatabase combines ideas from production rules, frames, and object-oriented representations, through the TSER modeling methodology. A detailed discussion of this approach to metadata representation can be found in recent paper [Hsu90?]. The TSER constructs are summarized in Table 1.

Table 1.a: TSER Functional Model (SER) constructs

CONSTRUCT	PRIMITIVES	DESCRIPTION
SUBJECT(SE)		
CONTEXT		

Note:

The rulebase model is designed essentially using the primitives provided at the structural (OER) modeling level of TSER; namely, the ENTITY and the three classes of RELATIONSHIP, plural-relationship (PR), functional-relationship (FR), and mandatory-relationship (MR). Since knowledge is represented and stored at the metadata level (i.e., in the metadatabase), the constructs comprising this rulebase model are referred to as meta-constructs; i.e., meta-entity and meta-relationships. For convenience, in this chapter and throughout the next one, the three classes of relationship are referred to as meta-PR, meta-FR, and meta-MR, respectively.

Modeling an application's knowledge starts with the identification of the basic objects (facts) and their relationships/interactions in the real-world of the application. Using the functional level constructs of TSER (i.e., SUBJECT and CONTEXT), an object is represented by the subject primitive and contains a set of data items as well as a set of operations defined on those items (i.e., encapsulation). Thus, a subject may be thought of as a frame (or object in an object-oriented representation) where its data items correspond to the slots of the frame. The operations correspond to methods and are specified in the form of production rules. Subjects may be grouped to form hierarchies of classes according to the generalization and aggregation properties [Smi77]. The interactions among various objects are also specified in terms of production rules and grouped according to their functionalities into contexts relating the appropriate subjects. These results (the information model) constitute the metadata of the application systems; they themselves are further abstracted into metadata models.

Based on this modeling concept, Figure 1 depicts the basic structure of the metadata model representing the application knowledge specified by these subjects and contexts. The model consists of four meta-entities and three meta-PR's. The two meta-entities **Subject** and **Context** represent respectively all subjects and contexts in the application

systems, respectively. The meta-entity **Item** represents the data contents of the subjects, while **Rule** represents those production rules included in both subjects and contexts. The fact that a subject encapsulates a set of rules and, at the same time, a rule may be involved in several subjects is reflected by the many-to-many meta-PR **Applies**. Similarly, the meta-PR **Contains** groups the rules into their appropriate contexts, while the meta-PR **Describes** links the data items to their corresponding subjects.

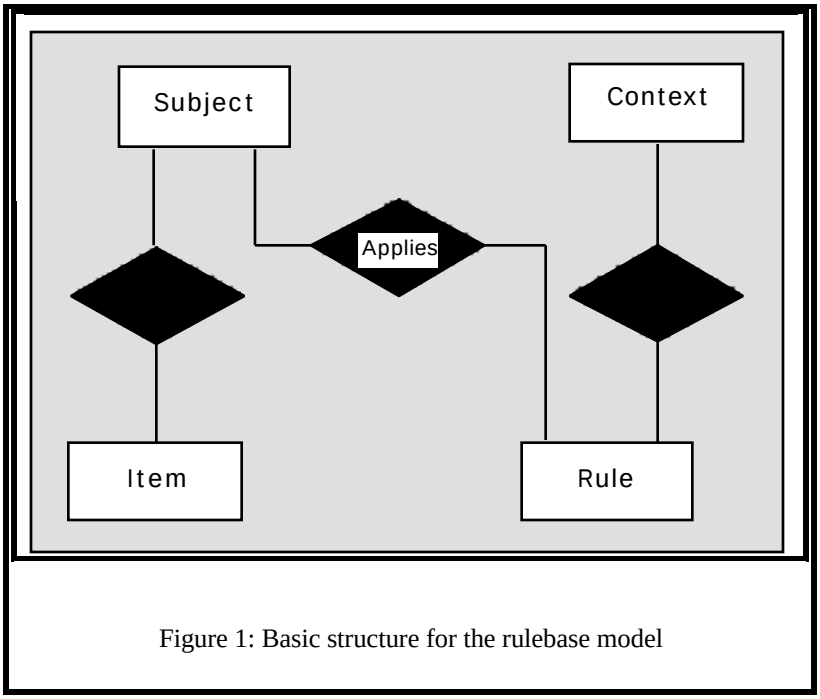


Figure 1: Basic structure for the rulebase model

The constructs discussed above provide an intuitive and abstract representation of knowledge. Details about their structures are discussed in the following sections towards developing the final rulebase model. This development is centered around the meta-entity **Rule**; which is decomposed further to derive a normalized structure representing the rules in terms of their basic components and their inter-relationships.

The basis of this derivation is the specific grammar of rules that we construct from general results in the field. It is shown as G1 and G2 below.

------> B

G1:

(1) <Rule> ::= IF <Condition> THEN [<Action>]$^{+}$

(2 <Condition> ::= <Expression>

(3) <Action> ::= <Declarative-Statement> | <Assignment-Statement>|
<Procedure-Call>

(4) <Assignment-Statement> ::= <Action-Ident> := <Evaluated-Fact>

(5) <Procedure-Call> ::= Procedure-name([<Parameter-List>]*)

G2:

(1) <Expression> ::= <Fact> <Operator> <Fact> | <Fact>

(2 <Fact> ::= <Evaluated-Fact> | <Declarative-Statement>

(3) <Evaluated-Fact> ::= <Simple-Fact> | <Composed-Fact>

(4) <Simple-Fact> ::= Constant | Item | <User-Ident> | <Action-Ident>

(5) <User-Ident> ::= Identifier

(6) <Action-Ident> ::= Identifier

(7) <Composed-Fact> ::= <Function-call> | <Expression>

(8) <Function-call> ::= Function-name([<Parameter-List>]*)

(9) <Parameter-List> ::= <Parameter>[, <Parameter-List>]*

(10) <Parameter> ::= <Fact>

(11) <Declarative-Statement> ::= <Simple-Fact> Verb <Simple-Fact>

3 Representation of Expressions in Rules

At the heart of the rulebase model is a structure using certain OER constructs to represent expressions which constitute the predicates of the rules. To illustrate the derivation of this structure, we consider first the representation of evaluated facts (see G2) using OER constructs

Simple Expressions: Facts

Facts constitute the basic units of rules and are represented in the rulebase model using a meta-entity called **Fact**. In this meta-entity, a fact is characterized with six meta-attributes: **Factid**, **Factname**, **Facttype**, **Factvalue**, **Valuetype**, and **Valueof**. The **Factid** meta-attribute is a system-generated identifier of the fact (i.e., the primary key of this meta-entity). **Factname** is the name with which item, identifier, and functional facts appear in the rules. The other facts (i.e., constant, composed expressions, and declarative statements) appear in the rules without names. **Facttype** is an integer number indicating the category of the fact as defined by grammar G2. **Factvalue** represents the value to be assigned to the fact and **Valuetype** indicates its data type (i.e., integer, real, character, etc.). The last meta-attribute, **Valueof**, is included to help the inference engine find or compute this value. It is: (1) an item identifier if the fact corresponds to a data item, (2) an expression identifier if the fact is a declarative statement or the result of an expression evaluation, or (3) a fact identifier (**Factid**) if the fact takes its value from another fact (i.e., bound by a rule's action). Table 2 summarizes the valid values (domains) of **Facttype** and **Valueof** meta-attributes.

Fact Represents	Facttype	Valueof
Constant	0	NULL
Data item	1	Item identifier
User identifier	2	NULL
Function value	3	NULL
Expression value	4	Expression identifier
Action identifier	5	Fact identifier
Declarative statement	6	Expression identifier

Table 2: Domains of Facttype and Valueof meta-attributes

Composed Expressions

Again the structure of the composed expressions is built from facts, as depicted in Figure 2. A composed expression consists of an operation between two facts (production (1) of G2); called the left and right operands of the operation. Accordingly, it is represented with the meta-entity **Expression** using three meta-attributes: **LeftFact**, **Operator**, and **RightFact**. **LeftFact** is a fact representing the value of the left operand, **Operator** is an arithmetic or logical operator, and **RightFact** is another fact that corresponds to the value of the right operand. A system-generated identifier (**Condid**) is also part of the structure of this meta-entity as its primary key. This meta-entity is also used to represent the declarative statements. The meta-attributes **LeftFact** and **RightFact** represent the two simple facts of the statement and **Operator** represents its verb.

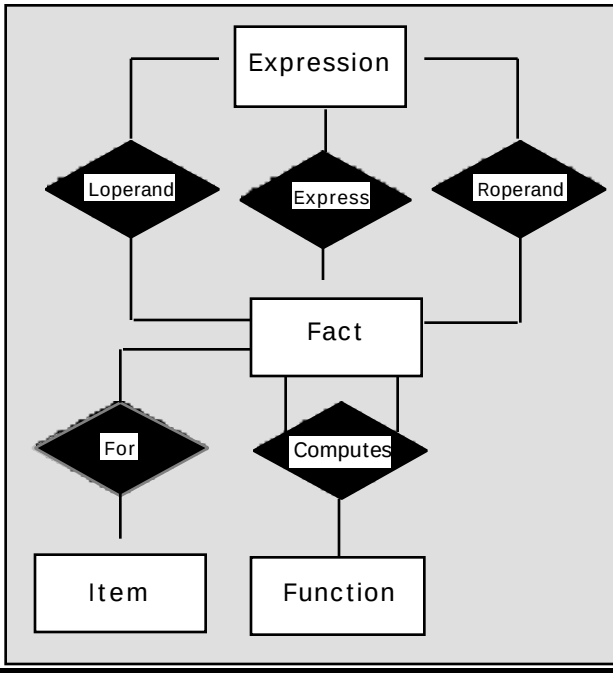


Figure 2 : OER representation of expressions

LeftFact and **RighFact** are synonyms of **Factid** which is the primary key of the meta-entity **Fact**. Hence, they are foreign keys in **Expression** that imply two existence dependency integrity constraints; i.e., a composed expression or declarative statement cannot exist without its left and right facts. These two constraints are represented by the meta-MR's **Loperand** and **Roperand**, with **Fact** as their owner meta-entity and **Expression** as their owned meta-entity. Similarly, the meta-MR **Express** represents another existence dependency between these two meta-entities, but in the opposite direction; i.e., **Expression** as the owner and **Fact** as the owned meta-entity. This double role of these two meta-entities is due to the recursive definition of expressions. While an expression contains two facts (which are foreign keys resulting in **Loperand** and **Roperand**), each fact in its own right can represent a composed expression (**Facttype** = 4) that is defined in the same way. In this case the meta-attribute **Valueof** of the meta-

entity **Fact** becomes a foreign key (since it represents the expression identifier; see Table 2) resulting in the meta-MR **Express**.

Valueof plays another role when its corresponding fact is a data item. It contains the item identifier corresponding to the meta-attribute **Itemcode** which is the primary key of the meta-entity **Item** (see Appendix A for all details of the data structures used). In this case, **Valueof** implies a referential integrity constraint that is represented by the meta-FR **For** between **Fact** and **Item** (which constitutes a unifying link between rules and data). Therefore, **Valueof** is a foreign key that implies either an existence dependency constraint (represented by the meta-MR **Express**) if the fact corresponds to a composed expression or a declarative statement, or a referential integrity constraint (meta-FR **For**) if the fact is a data item. The difference between the two boils down to the type of semantics adopted in each case. In the former, the fact cannot exist without the expression it represents, while in the latter the fact becomes a run-time identifier fact if its corresponding data item is deleted. That is, only its **Facttype** value needs to be modified to "2" (for run-time identifiers) and its corresponding **Valueof** meta-attribute needs to be set to NULL so that it does not reference the deleted data item.

Finally, the meta-PR **Computes** in Figure 2 connects functional facts (representing values returned by function execution; see Table 2) to their respective functions. It consists of the meta-attributes **Factid**, **Functid**, **Parid**, and **Parorder**. **Factid** is the identifier of the functional fact and **Functid** identifies the function itself. **Parid** (defined on the same domain as **Factid**) and **Parorder** correspond to the parameters of the function and their relative order during the function call, respectively. The meta-entity **Function** represents metadata about the functions called by the rules; their contents are stored separately (see [Bou91] for storage details).

Example 1:

To illustrate the representation of expressions using the above model, consider three expressions E1, E2, and E3 as given below. The first expression (E1) is a composed expression whose left and right facts are declarative statements, where "ismadeof" and "ismadeby" are two acceptable operators (in this case verbs). The second expression (E2) involves a fact whose value is computed by a function and the third one (E3) contains composed (sub-)expressions.

- E1: (PR ismadeof P) AND (P ismadeby M)
  (Note: PR, P, and M are facts corresponding respectively to the data items Product-id, Part-id, and Machine-id)
- E2: Delta = 0
  (Note: Delta corresponds to the value returned by the function ComputeDelta(A, B, C); A, B, and C are identifiers)
- E3: (- B) / (2 * A)

E1 is represented in Table 3 and both E2 and E3 in Table 4. E1 involves five facts represented by the meta-entity **Fact**. The first three (F1, F2, and F3) correspond to the three persistent data items represented by **Item**, while the other two facts (F4 and F5) correspond to the left and right declarative statements (as a whole) of the expression. For reading clarity of this example, the last two facts have been given the names "PRofP" and "PbyM", respectively. The contents of these two facts (each in its own right is an expression) are represented as the first two instances in the meta-entity **Expression**, with **Valueof** and **Condid** providing the connections between **Fact** and **Expression** (i.e., through matching values of C1 and C2). The last instance of **Expression** corresponds to the expression E1 itself.

Fact

Factid	Factname	Facttype	Factvalue	Valuetype	Valueof
F1	PR	1	?	STRING	I1
F2	P	1	?	STRING	I2
F3	M	1	?	STRING	I3
F4	PRofP	6	?	BOOLEAN	C1
F5	PbyM	6	?	BOOLEAN	C2

Expression

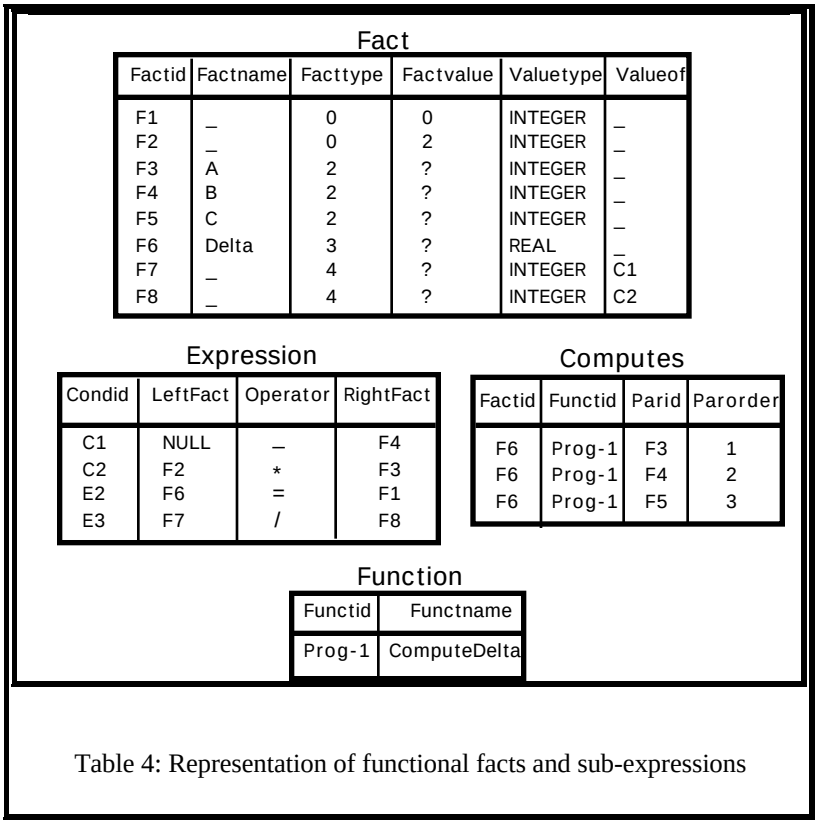
Condid	LeftFact	Operator	RightFact
C1	F1	ismadeof	F2
C2	F2	ismadeby	F3
E1	F4	AND	F5

Item

Itemcode	Itemname
I1	Product-id
I2	Part-id
I3	Machine-id

Table 3: Representation of declarative statements

The representation of expressions E2 and E3 involves **Expression**, **Fact**, and **Function** meta-entities and the meta-PR **Computes**. Four categories of facts are identified in these expressions: constant facts (**Facttype** = 0), identifiers (**Facttype** = 2), function values (**Facttype** = 3), and expression values (**Facttype** = 4). The simplest facts are constants “0” (in E2) and “2” (in E3), which correspond to the first two instances (i.e., F1 and F2) of **Fact**. The next four are user-supplied identifiers A, B, and C and the function value Delta resulting from the execution of the function ComputeDelta; which is represented in the meta-entity **Function**.



Fact

Factid	Factname	Facttype	Factvalue	Valuetype	Valueof
F1	_	0	0	INTEGER	_
F2	_	0	2	INTEGER	_
F3	A	2	?	INTEGER	_
F4	B	2	?	INTEGER	_
F5	C	2	?	INTEGER	_
F6	Delta	3	?	REAL	_
F7	_	4	?	INTEGER	C1
F8	_	4	?	INTEGER	C2

Expression

Condid	LeftFact	Operator	RightFact
C1	NULL	_	F4
C2	F2	*	F3
E2	F6	=	F1
E3	F7	/	F8

Computes

Factid	Functid	Parid	Parorder
F6	Prog-1	F3	1
F6	Prog-1	F4	2
F6	Prog-1	F5	3

Function

Functid	Functname
Prog-1	ComputeDelta

Table 4: Representation of functional facts and sub-expressions

The sub-expressions (- B) and (2 * A) of the expression E3 are represented as facts F7 and F8, respectively. Again, the meta-attribute **Valueof** in each instance of these facts contains the identifier of the corresponding sub-expression.

Finally, information about the function called in E2 (i.e., its name and other information that helps locating the function and eventually executing it) is represented by the instance in **Function** linking to three instances in **Computes**. These instances specify the parameters of the function and their relative order as facts F3 (A), F4 (B), and F5 (C).

The connection between facts and rules through this representation of expressions is the final step in the rulebase modeling. Essentially, this connection relates conditions of rules to expressions and indicates those fact values that result from action executions of rules, as discussed in the next section.

4. The Rulebase Model

A production rule is formally structured in this research by grammar G1. It consists of a condition and one or more actions; where the actions of the rule are executed (i.e., the rule is fired) whenever its condition is satisfied. The condition is a logical expression and, hence, can be defined by grammar G2 given in the previous section.

Each action can be a declarative statement, an assignment statement, or a procedure call (production (3) of G1). When an action is a declarative statement, it corresponds to a declarative-statement fact in the meta-entity **Fact** and, when executed, establishes the truth value of this fact (i.e., it sets the meta-attribute **Factvalue** of this fact to TRUE). The execution of actions that are assignment statements causes the right hand side fact to be bound to a value, which is then assigned to the identifier fact. For this reason, in Table 2, the facts of type "action identifier" (**Facttype** = 5) use the meta-attribute **Factid** from evaluated facts for their **Valueof**. The third type of action uses facts for procedures' parameters. Therefore, actions in the rules provide the mechanism for controlling the interaction between declarative statements and procedural knowledge.

Normalization of Rules:

So far, we have represented rules, including their conditions and actions, with a single meta-entity (**Rule** in Figure 1). However, when defined using grammar G1, the meta-entity **Rule** needs to be normalized. First, its "condition" element contains the same meta-attributes of the meta-entity **Expression** (thus forming an embedded group in it). Second, it comprises repeating groups since a rule can have many actions. Therefore, following the TSER mapping algorithms, this meta-entity is normalized by decomposing it into three meta-entities: **Rule**, **Condition**, and **Action**; as shown in Figure 3.

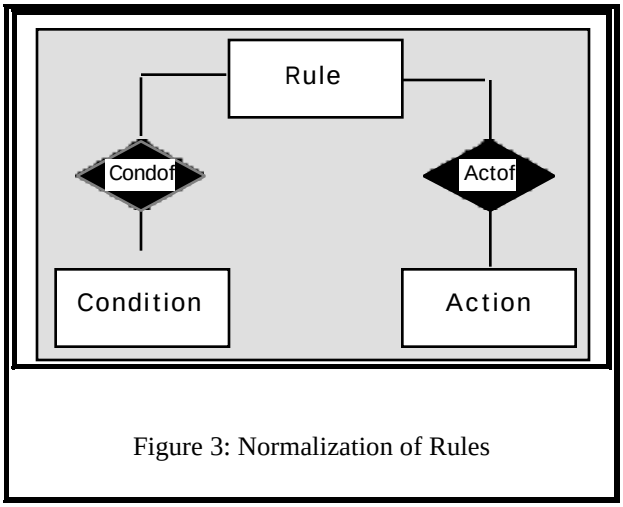


Figure 3: Normalization of Rules

The actions are linked to the proper rules by means of the meta-PR **Actof**, while the condition is linked through the meta-FR **Condof**. The difference in such linking is due to the removal of repeating groups versus the transitive functional dependencies (embedded meta-attributes). A rule can have many actions (see production (1) of G1), each of which can be involved in one or more rules, thereby establishing the many-to-many relationship (**Actof**). The condition of a rule, on the other hand, is single and can be involved in many rules, and hence the many-to-one relationship **Condof** (a condition may have a set of sub-conditions glued together through the logical operators AND/OR).

Condition:

A condition is an expression. Therefore, the meta-entity **Condition** is structurally equivalent to the meta-entity **Expression** (Figure 2) described in the previous section. In other words, the meta-entity **Condition** contains the meta-attributes **Condid**, **LeftFact**, **Operator**, and **RightFact**.

Action:

Actions of the rules are represented with the meta-entity **Action** using three meta-attributes: **Actid**, **Acttype**, and **Factid**. **Actid** is a system-generated identifier of actions

and **Acttype** indicates the action type (i.e., assignment statement, declarative statement, or a procedural action). The meta-attribute **Factid** identifies the fact corresponding to a declarative statement or the fact bound by an assignment statement (i.e., the left hand side fact of the statement). It is a foreign key implying a referential integrity constraint that is represented in Figure 4 below by the meta-FR **Bind-Fact**.

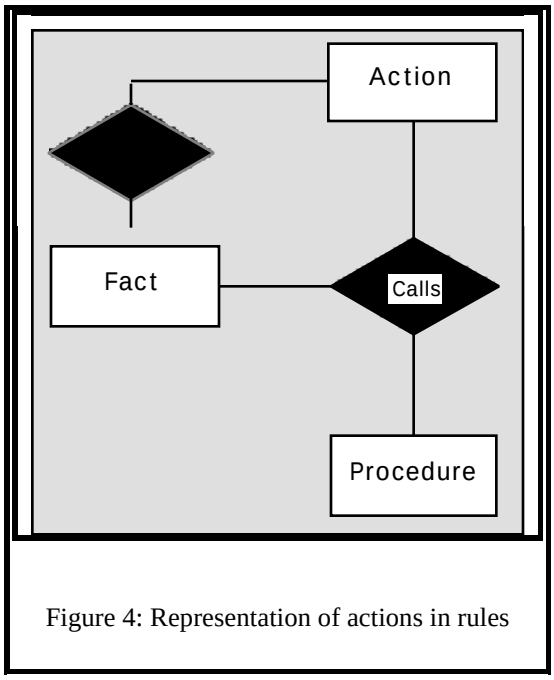


Figure 4: Representation of actions in rules

The meta-PR **Calls** connects the procedural actions to their corresponding procedures using the **Actid**, **Procid**, **Parid**, and **Parorder** meta-attributes. **Actid** and **Procid** are the identifiers of these actions and their associated procedures, respectively, while **Parid** identifies those facts used as the procedure's parameters and **Parorder** indicates their relative order in the call. The meta-entity **Procedure** represents information about these procedures. Its structure is functionally equivalent to that of the meta-entity **Function** (Figure 2).

Rule:

With this representation, the meta-entity **Rule** in Figure 3 contains **Rname** (rule name identifying the rules), **Rtype** (rule type; e.g., conversion, operating, etc.), **Descript** (rule description), and **Condid** (as a foreign key connecting the rules to their conditions).

It also includes other meta-attributes such as **Addedby** and **Dateadded** for audit trail information about the rules.

Consolidation

The above OER constructs representing the different components of the rulebase are consolidated into an overall OER model. Figures 1 and 3 are connected through the meta-entity **Rule**. The result is linked to the constructs of Figure 2 by replacing the **Expression** meta-entity with **Condition** (which, as discussed earlier are equivalent structurally and logically) and merging the two appearances of the **Item** meta-entity. Finally, the constructs of Figure 4 are incorporated into the model through the **Action** meta-entity, with **Function** and **Procedure** meta-entities (which are identical in contents) replaced by the meta-entity **Routine**.

The result of this consolidation is shown in Figure 5 and constitutes a generic structure for the rulebase model. The next section illustrates the feasibility of representing and storing different types of rules with this generic rulebase modeling approach.

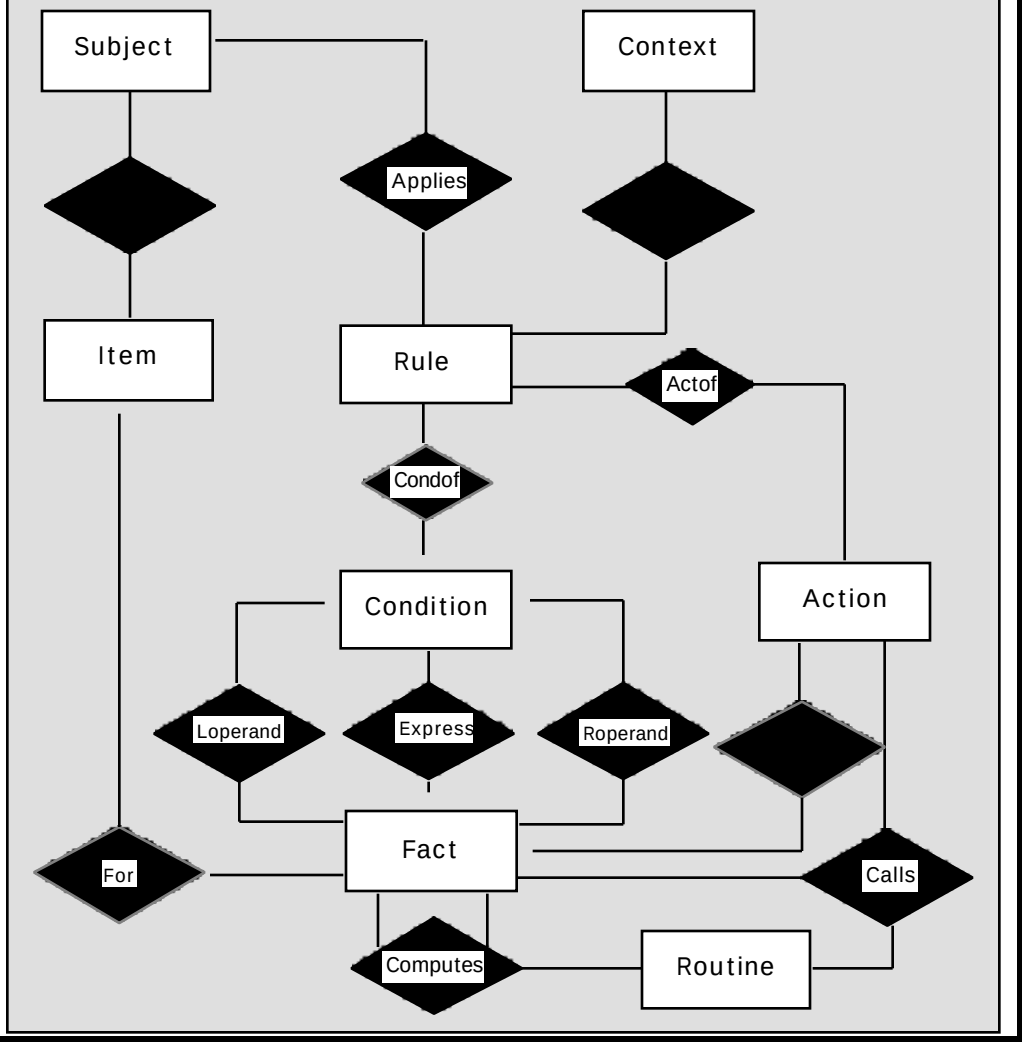


Figure 5: The Rulebase Model

5. Illustration and Discussion

The above rulebase model is illustrated with two examples, one using production rules and the other mathematical equations. In each example, we first present the information resources and expressions involved, then discuss how they are represented with the rulebase model.

Example 2: Representation of Production Rules

The sample database shown in Table 5 contains information about product, part, and machine entities and the relationships among them in a manufacturing environment. Each product is made of a set of parts (i.e., comprising its bill of materials) and each part may be recursively composed of sub-parts. These two relationships are represented by BILLMAT and PARTOF relations, respectively. The fact that a certain machine is used to produce a specific part is captured by the relation MAKES.

PART

Part-id	Partname	-----
P1	Engine	-----
P2	Chasis	-----
P3	Wheel	-----
P4	Cylinder	-----
P5	Tire	-----
P6	Nut	-----
P7	Carburator	-----

PRODUCT

Product-id	Prodname	---
P104	Auto	---
P200	Truck	---

MACHINE

Machine-id	Machname	---
M1	Milling	---
M2	Assembly	---

PARTOF

Subpart-id	Part-id
P4	P1
P5	P3
P6	P3
P7	P4

BILLMAT

Product-id	Part-id
PR104	P1
PR104	P2
PR104	P3

MAKES

Machine-id	Part-id
M1	P7
M2	P1

Table 5: A Sample of Product Database

The Rules:

The four production rules given in Table 6 encode the structural knowledge underlying the schema of the product database (Table 5).

- R1: IF (PR ismadeof P) AND (P ismadeby M)
  THEN (PR uses M)

- R2: IF Istuple(PR, P, “BILLMAT”)
  THEN (PR ismadeof P)
- R3: IF Istuple(M, P, "MAKES")
  THEN (P ismadeby M)
- R4: IF Istuple(M, Q, "MAKES") AND Ispartof(Q, P)
  THEN (P ismadeby M)

Table 6: An example of production rules

Basically, these rules operate on the data items in this database to infer whether a given machine is used in manufacturing a product. This fact is deduced by rule R1, while the other three rules are mainly used to establish the left fact (i.e., “PR ismadeof P”) and the right fact (i.e., “P ismadeby M”) in the condition of this rule (R1). The left fact is inferred by rule R2, while the right fact can be established either by rule R3 or rule R4; depending on whether the machine in question makes directly or indirectly (i.e., a component of) a part in the product’s bill of materials.

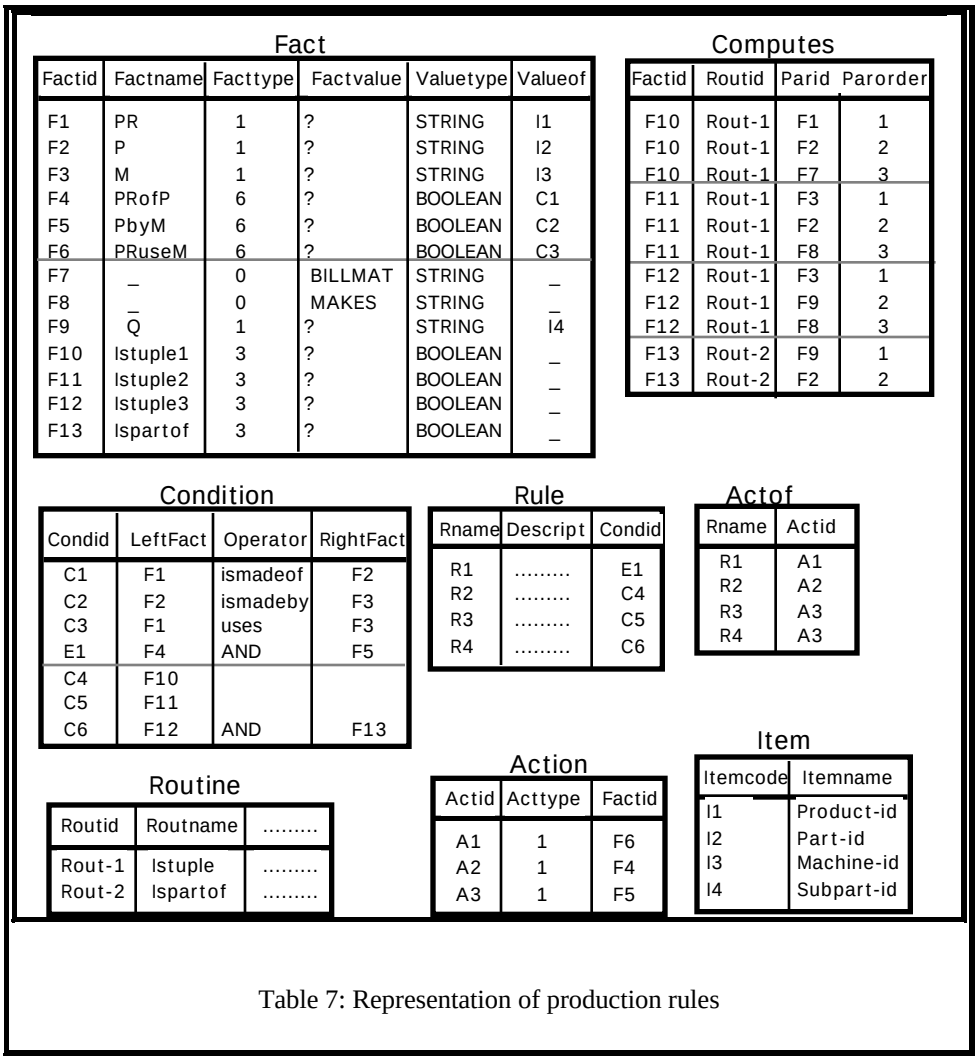
The functions “Istuple” and “Ispartof” are used by the rules to access the database. The former checks whether the pair formed by its first two arguments is an instance in the relation specified by its third argument, while the latter (recursively) searches through the instances of the relation PARTOF to find out if part P (second argument) has a sub-part Q (first argument).

The Representation:

The complete representation of the four rules above is depicted in Table 7. The first six facts in the meta-entity **Fact**, F1-F6, are extracted from rule R1 in a manner explained in Example 1; the remaining are from rules R2, R3, and R4. Note that (1) BILLMAT (F7) and MAKES (F8) are relation names in Table 5, (2) the variable Q (F9) corresponds to the data item Subpart-id (which is represented in **Item**), and (3) facts F10-F13 are the four function calls that appear in the conditions of these rules. For reading clarity, the facts for

the three calls to the function “Istuple” are given respectively the names Istuple1, Istuple2, and Istuple3 in the order of their appearance in the rules.

For each of the function-call facts, a group of instances is included in the meta-PR **Computes** that connects this fact to its corresponding function. For example, the first group in this meta-PR shows that fact F10 gets its value from the call to the function identified by “Rout-1” with parameters: F1 (representing PR), F2 (P), and F7 (“BILLMAT”). Thus, it corresponds to the call Istuple(PR, P, “BILLMAT”) represented in the meta-entity **Routine**. The other groups can be interpreted in the same way.



Fact

Factid	Factname	Facttype	Factvalue	Valuetype	Valueof
F1	PR	1	?	STRING	I1
F2	P	1	?	STRING	I2
F3	M	1	?	STRING	I3
F4	PRofP	6	?	BOOLEAN	C1
F5	PbyM	6	?	BOOLEAN	C2
F6	PRuseM	6	?	BOOLEAN	C3
F7	_	0	BILLMAT	STRING	_
F8	_	0	MAKES	STRING	_
F9	Q	1	?	STRING	I4
F10	Istuple1	3	?	BOOLEAN	_
F11	Istuple2	3	?	BOOLEAN	_
F12	Istuple3	3	?	BOOLEAN	_
F13	Ispartof	3	?	BOOLEAN	_

Computes

Factid	Routid	Parid	Parorder
F10	Rout-1	F1	1
F10	Rout-1	F2	2
F10	Rout-1	F7	3
F11	Rout-1	F3	1
F11	Rout-1	F2	2
F11	Rout-1	F8	3
F12	Rout-1	F3	1
F12	Rout-1	F9	2
F12	Rout-1	F8	3
F13	Rout-2	F9	1
F13	Rout-2	F2	2

Condition

Condid	LeftFact	Operator	RightFact
C1	F1	ismadeof	F2
C2	F2	ismadeby	F3
C3	F1	uses	F3
E1	F4	AND	F5
C4	F10		
C5	F11		
C6	F12	AND	F13

Rule

Rname	Descript	Condid
R1	.....	E1
R2	.....	C4
R3	.....	C5
R4	.....	C6

Actof

Rname	Actid
R1	A1
R2	A2
R3	A3
R4	A3

Routine

Routid	Routname	.....
Rout-1	Istuple	.....
Rout-2	Ispartof	.....

Action

Actid	Acttype	Factid
A1	1	F6
A2	1	F4
A3	1	F5

Item

Itemcode	Itemname
I1	Product-id
I2	Part-id
I3	Machine-id
I4	Subpart-id

Table 7: Representation of production rules

The rules are represented in **Rule** with the identifiers of their respective condition. The function calls in the conditions of rules R2 and R3 are identified by facts F10 and F11. Thus, to represent each of these conditions, it is sufficient to include the fact representing its function call as the meta-attribute **LeftFact** (as shown in instances C4 and C5 of **Condition**). The condition of rule R4 is represented by the instance C6 as an AND operation between facts F12 and F13. The two actions A2 and A3 of rules R2, R3, and R4 (action A3 is shared by rules R3 and R4 as indicated in **Actof**) are of type declarative statement and are represented in the same way as action A1.

This example shows how the rulebase model can be used to represent production rules and to specify a suitable structure for storing them in a database form. The next example shows how mathematical equations can be handled in the same way.

Example 3:

The three rules shown in Table 8 are used to solve a second degree equation with one variable (i.e., $AX^2+BX+C=0$; where the values of A, B, and C are known). Based on the value of the variable Delta (computed by the function "ComputeDelta" using the formula given in the table), an equation either has no real solution (Delta < 0), one real solution (Delta = 0), or two real solutions (Delta > 0).

- Delta = B^2 - 4*A*C
- Qualify = $\sqrt{\text{Delta}}$ / (2*A)

- R1: IF Delta < 0
  Then ComplexRoots(Delta, A, B)
- R2: IF Delta = 0
  Then X := (- B) / (2*A)
- R3: IF (Delta > 0)
  Then Y := (- B) / (2*A) + Qualify

Z := (- B) / (2*A) - Qualify

Table 8: An example of mathematical rules

Table 9 shows the representation of these rules. The meta-PR **Calls** connects the action of rule R1 to the procedure ComplexRoots. Its instances represent the procedure call in the same way as function calls are represented in **Computes**. The interpretation of the other constructs in this table has been discussed in the previous examples.

Fact

Factid	Factname	Facttype	Factvalue	Valuetype	Valueof
F1	-	0	0	INTEGER	-
F2	-	0	2	INTEGER	-
F3	A	2	?	INTEGER	-
F4	B	2	?	INTEGER	-
F5	C	2	?	INTEGER	-
F6	Delta	3	?	REAL	-
F7	-	4	?	INTEGER	C1
F8	-	4	?	INTEGER	C2
F9	-	4	?	REAL	E3
F10	X	5	?	REAL	F9
F11	Qualify	3	?	REAL	-
F12	-	4	?	REAL	C3
F13	-	4	?	REAL	C4
F14	Y	5	?	REAL	F12
F15	Z	5	?	REAL	F13

Computes

Factid	Routid	Parid	Parorder
F6	Rout-1	F3	1
F6	Rout-1	F4	2
F6	Rout-1	F5	3
F11	Rout-2	F6	1
F11	Rout-2	F3	2

Calls

Actid	Procid	Parid	Parorder
A1	Rout-3	F6	1
A1	Rout-3	F3	2
A1	Rout-3	F4	3

Routine

Routid	Routname	.....
Rout-1	ComputeDelta	.....
Rout-2	RootQualifier	.....
Rout-3	ComplexRoots	.....

Actof

Rname	Actid
R1	A1
R2	A2
R3	A3
R3	A4

Condition

Condid	LeftFact	Operator	RightFact
C1	NULL	-	F4
C2	F2	*	F3
E2	F6	=	F1
E3	F7	/	F8
C3	F9	+	F11
C4	F9	-	F11
C5	F6	<	F1
C6	F6	>	F1

Rule

Rname	Descript	Condid
R1	.....	C5
R2	.....	E2
R3	.....	C6

Action

Actid	Acttype	Factid
A1	2	-
A2	0	F10
A3	0	F14
A4	0	F15

Table 9 Representation of mathematical rules

6. Unique Properties of the Rulebase Model

The need for the rulebase model in KBMS --i.e., integration of data and knowledge-- can be analyzed at two levels. Practically, there are two main design approaches for achieving such an integration when large volume of data and knowledge are involved; loose coupling and tight coupling. The loose coupling approach (see [Bro86] for examples) consists of connecting an existing expert system with a DBMS through communication links. Since rules are stored and managed by the expert system component, the integration is addressed only at a physical level and would inevitably hinder the performance of the system.

The tight coupling strategy, on the other hand, calls for an environment where rules and data are stored and managed by the same component, which tends to be a DBMS not only because expert systems are not poised for data management, but also since data is the common basic element of both sides. However, in order for any DBMS to store rules, a conceptual model of rules is essential. This is where the rulebase model becomes physically necessary. It not only provides a global conceptual representation of rules, but also produces a rulebase schema for storing and managing rules using a conventional DBMS.

At the second level, however, the significance of the rulebase model is more general and addresses directly some of the basic limitations of the previous expert systems methods mentioned above. The rulebase model (1) supports rule independence and (2) integrates rules with data. Rule independence is needed whenever rules are to be shared or re-used in different systems or views (e.g., user-defined subjects, objects, and frames in queries). The integration of data and rules involves capturing the inter-relationships among rules and relating the contents of these rules to data objects. To achieve this goal, however, rules must be decomposed and modeled through their basic primitives (i.e., facts and logic).

This is the same idea of data modeling methodologies: building complex data structures and data semantics from data items and their associations. Furthermore, to facilitate access and manipulation of rules (especially in a multiple system environment such as the metadatabase), the modeling of rules must be global and at the same level as data models, as opposed to physically encapsulating rules within objects as in object-oriented models. Therefore, by modeling rules globally from their facts and relating these facts to data items (through the meta-FR **For**), the rulebase model integrates rules with data in the metadatabase at a conceptual level.

Recently, the need for global conceptual modeling of rules has been recognized by many researchers and practitioners. Research in this direction includes the Relational Production Language (RPL) [Del89] and the Data Intensive Production System (DIPS) [Sel88]. RPL is an SQL-based forward chaining language that has an underlying relational model for rules. However, this model is limited to representing only facts and leaves much of the logic to application programs. The conditions of rules are specified as SQL queries to the database containing these facts and the actions are limited to SQL data manipulation operations on this database. DIPS is another approach that employs the relational technology for modeling rules, but its model is also limited. Only facts and conditions of rules are represented in the model. The actions are specified through ad-hoc queries to the database containing facts and conditions.

The rulebase model exceeds these ideas and includes the full contents of rules (i.e., facts, conditions, and actions) using TSER which allows a more abstract, semantical representation of rules than the relational approach. Like object-oriented models, it also employs the concept of encapsulating rules with objects (i.e., subjects and contexts in TSER). Unlike object-oriented models, however, this encapsulation in the rulebase is virtual not physical; i.e., the rules are modeled globally and connected logically to their

respective subjects and contexts through the meta-relations **Applies** and **Contains**, respectively.

When compared to previous knowledge representation methods, the rulebase model offers several advantages that are summarized below:

(1) It provides a well-structured approach to knowledge representation that enables mixing rules with procedures. This structure is based on TSER which defines rigorous integrity controls for rules.

(2) It allows implementing large knowledge bases in secondary memory.

(3) It enables a tight integration and a unified management of rules and data in the metadatabase using conventional database technology. Thus standard and established features of existing database systems such as update integrity, concurrency, and sharing can be applied to rules.

(4) It constitutes a natural organization of rules; i.e., rules are logically grouped into subjects and contexts which, in turn, permit efficient clustering of rules for searching and processing.

(5) It supports storing rules in a pre-compiled form that allows efficient evaluation and management of rules.

(6) It implicitly but definitively represents rule chaining through common facts and integrity constraints.

7. Further Remarks

The rulebase model is developed to achieve a specific goal: integrate contextual knowledge (in the form of production rules) with data models to support effective management of (a large volume of) rules. This need stems from the requirements of the metadatabase approach; by satisfying them, however, the rulebase model also establishes a

class of generic properties that distinguish itself from previous methods of rules representation and management.

These requirements include the ability to (1) facilitate rules sharing among applications of all functional systems across an enterprise, (2) represent integrity constraints and other dynamic relationships among data objects throughout, (3) provide simple procedures that manipulate (add, delete, modify) rules along with any other data objects, and (4) support analysis and creation of new views for global query and other applications. All these requirements are beyond the scope and capabilities of conventional rule-based systems (or expert systems).

Specifically, these systems take an unstructured approach to rules representation [Leu90]. There is no underlying model for representing the inter-relationships of rules or controlling their inconsistencies. Thus, rules within one system cannot be shared or cross-referenced by other systems. Moreover, rules within these systems are usually stored in main memory in their convoluted source (text) form [Boc89], making their maintenance and management an overwhelming problem for large and complex systems. Finally, the data objects underlying these rules are neither abstracted independently nor calibrated to data models for effective management. For example, a simple query such as finding the data objects involved in a rule is a very difficult task since it requires extensive parsing and interpreting the text of rules.

At the heart of this rulebase model is the representation design that the rules are identified through the basic units of a knowledge base: facts. The conditions and actions of the rules are built around those facts and connected together by an appropriate set of relationships that portray the rule semantics. Therefore, a rule is stored only once and can be shared by as many subjects and contexts (and hence systems) as called for by the

organization. This sharing is even extended to the facts, conditions, and actions among the rules.

As such, this rulebase model provides a framework for consolidating different knowledge-based systems that results in several tangible benefits for the host enterprise. These include the reduction of knowledge redundancy, an increase in modular system design and knowledge independence, and a high degree of control over of the enterprise knowledge resources. All of them contribute to the generic field of Knowledgebase Management System as well as the particular areas of data and knowledge integration in computerized manufacturing.

Furthermore, the model promises efficient and effective rulebase processing by virtue of (1) allowing full relational capability, (2) providing integrity control, and (3) facilitating rulebase search. The first two promises are derived immediately from the TSER modeling approach it employs (e.g., an entity or relationship is characterized with a relational data structure plus definitive typing rules). The third is due to its structure. For instance, finding appropriate rules in the chaining operation of a rulebase processor can simply performed by joining **Fact** with **Action** on the meta-attribute **Factid**, which is common to both meta-entities, using **Bind-Fact** (see Figure 5). With this property and the notion of grouping the rules according to the subjects and contexts they pertain to, an efficient rule processor system can be built based on this structure. The search algorithms used in the reasoning process can be bound to search within the same group of rules first, rather than selecting the rules randomly, or according to some predetermined selection method.

A prototype rulebase is developed as part of a metadatabase system at Rensselaer's Computer-Integrated Manufacturing Program. This prototype includes both global data management rules and event-triggered operating rules for a heterogeneous multiple-system

environment consisting of: an order entry system using Rdb, a shop floor control system based on Oracle, and a process planning system developed in dBase with Object-Oriented typing design; all of which are integrated through the metadatabase. Major works for the future include: (1) devising new algorithms utilizing the unique properties of the rulebase model, (2) developing linkage of the rulebase model with representative knowledge representation methods and systems towards a design of KBMS, and (3) applying the rulebase capabilities to support information integration.