Constraints for Knowledge Maintenance

John Debenham

Key Centre for Advanced Computing Sciences
University of Technology, Sydney
PO Box 123, NSW 2007, Australia
debenham@socs.uts.edu.au

Abstract

Constraints are used to manage the integrity of knowledge. Knowledge-based systems applications are frequently constructed as a knowledge base, or rule base, component coupled with a relational database component. The most expensive of these two components to build and maintain is typically the knowledge base. Constraints, which play a key role in database management, are seldom mentioned in connection with knowledge. One class of knowledge constraints protect the integrity of knowledge bases during maintenance by preventing the introduction of update anomalies. Another class of knowledge constraints contributes to the efficiency of the maintenance procedure. The efficiency of the maintenance procedure is increased further if the knowledge in the knowledge base has been normalised. An experimental knowledge-based systems design and maintenance tool which incorporates this approach has been built and trialed in a commercial environment.

Executive Summary

Knowledge-based systems applications are frequently constructed as a knowledge base, or rule base, component containing the 'knowledge' which is coupled with a relational database component containing the 'information' (Debenham 1996b) (Gray 1989). The more expensive of these two components to build and maintain is typically the knowledge base (Tayar 1993). Constraints, which play a key role in database management, are applied both to protect the integrity of the knowledge and to improve the efficiency of the knowledge base maintenance procedure (Debenham & Devedžić 1996a).

The cost of managing system integrity increases with the degree of "volatility" of a system (Devedžić 1996). *Volatile* systems are systems which are subject to a high rate of change. Constraints are used to manage the integrity of the knowledge-based systems in which the knowledge base is volatile (Coenen & Bench-Capon 1992). This approach employs constraints for two distinct purposes:

- constraints protect the integrity of the knowledge base during maintenance by preventing the introduction of update anomalies (Kang Gambetta & Compton 1996) (these are called *pragmatic constraints*), and
- constraints contribute to the efficiency of the maintenance procedure (these are called *referential constraints*).

Constraints are an integral part of the "conceptual model". The *conceptual model* is a complete representation of the knowledge required by the system; it specifies *how* the system will do what it is required to do (Debenham 1996d). The conceptual model consists of two parts. The first part is a representation of all things in the application as "items" (Debenham 1996b). The second part is a "coupling map" which supports the maintenance procedure (Debenham 1995). Items contain pragmatic constraints. Referential constraints simplify the structure of the coupling map. A tool which incorporates this approach has been built and trialed in a commercial environment.

The terms 'data', 'information' and 'knowledge' are used here in a rather idiosyncratic sense. The *data* in an application are those things which are taken as the fundamental, indivisible things in that application; data things can be represented as simple constants or variables. The *information* is those things which are "implicit" associations between data things. An *implicit* association is one that has no succinct, computable representation. Information things can be represented as tuples or relations. The *knowledge* is those things which are "explicit" associations between information things or data things. An *explicit* association is one that has a succinct, computable representation. Knowledge things can be represented either as programs in an imperative language or as rules in a declarative language.

The conceptual model is not a complete system specification. The conceptual model does not contain details of which information items should be stored as relations in the relational database. The conceptual model does not contain details of how the knowledge items should be used to derive the values of those information items which are not physically stored as relations. The *internal model* is derived from the conceptual model by including both details of which information items should be stored as relations in the relational database, and details of how the knowledge items should be used to derive the values of those information items which are not physically stored as relations (Debenham & Devedžić 1996a). Once the internal model has been derived, referential constraints may

be applied to the items in the conceptual model. These referential constraints simplify the coupling map, and thus improve the efficiency of the maintenance procedure.

Items have a uniform format no matter whether they represent data, information or knowledge. The key to this uniform representation is the way in which the "meaning" of an item, called its *semantics*, is specified. The semantics of an item is a function which *recognises* the members of the "value set" of that item. The *value set* of an information item is the set of tuples which are associated with a relational implementation of that item. Knowledge items, including complex, recursive knowledge items, have *value sets* too (Debenham 1996b). For example, the item which represents the rule "the sale price of parts is the cost price marked up by a universal mark-up factor" could have a value set as shown in Figure 1. Items incorporate two distinct classes of pragmatic constraints.

Items are expressed in terms of their "components". For example, the knowledge item which represents the rule for marking-up the price of car spare parts will be expressed in terms of car spare parts. Also the knowledge item which represents the rule for marking-up the price of bike spare parts will be expressed in terms of bike spare parts. Items are thus unable to express the essence of the "mark up" rule. Objects are item building operators. Objects are not expressed in terms of particular components. Items and objects have a similar structure. The semantics of an object is a function which *recognises* the members of the "value set" of any item instance of that object operator. Objects incorporate two distinct classes of pragmatic constraints.

Pragmatic constraints apply equally to knowledge, information and data. A taxonomy of pragmatic constraints is:

- constraints which are attached to each item (these are called the *item constraints*), these are:
  - the item *value constraints* which are constraints on the individual members of an item's value set, and
  - the item *set constraints* which are constraints on the structure of an item's value set. Set constraints include:
    - *cardinality constraints*, denoted by "Card", which constrain the size of the value set;
    - *universal constraints*, denoted by "Uni", which generalise database universal constraints, and
    - *candidate constraints*, denoted by "Can", which constrain the functional dependencies in an item and generalise database key constraints.
- constraints which are attached to the conceptual model itself (these are called the *model constraints*).

Referential constraints contribute to the efficiency of the maintenance procedure. Referential constraints apply equally to knowledge, information and data items. Referential constraints complicate the item to which they are applied and should thus only be applied to items of low volatility.

A major collaborative research project between the University of Technology, Sydney and the CSIRO Division of Information Technology has addressed the effective maintenance of knowledge-based systems. The early results of this project are summarised in (Debenham 1989). Recent work in this project has focused on two issues. The first issue is the development of a unified framework for conceptual modelling in which the "data", "information" and "knowledge" in the application can all be represented entirely as "items" in a single formalism (Debenham & Devedžić 1996b). The second issue is the development of classes of constraints for knowledge which can protect the knowledge base effectively against the introduction of update anomalies. Early results on the development of knowledge constraints were reported in (Debenham 1989); more recent results are reported here. A key product of this collaborative research project has been the development of a complete methodology for the management of knowledge-based systems. This methodology is supported by a Computer Assisted Knowledge Engineering tool, or CAKE tool, called "The Knowledge Analyst's Assistant". An experimental version of this tool has been constructed (Debenham 1989) and has been trialed in a commercial environment.

Pragmatic Constraints

Pragmatic constraints are an integral part of the "conceptual model". The *conceptual model* consists of both a representation of the things in the applications as "items", and a coupling map. The items in the conceptual model may *either* be specified explicitly *or* be specified implicitly as a sequence of object operators applied to a set of basic data items.

[part/sale-price, part/cost-price, mark-up]

part/sale-price		*part/cost-price*		*mark-up*
part-number	dollar-amount	part-number	dollar-amount	factor
1234	1.48	1234	1.23	1.2
2468	2.81	2468	2.34	1.2
3579	4.14	3579	3.45	1.2
8642	5.47	8642	4.56	1.2
7531	6.80	7531	5.67	1.2
1470	8.14	1470	6.78	1.2

Figure 1 Value set of the item *[part/sale-price, part/cost-price, mark-up]*

name		item name
name1	*name2*	item components
x	y	dummy variables
meaning of item		item semantics
constraints on values		item value constraints
set constraints		item set constraints

part/cost-price	
part	*cost-price*
x	y
costs(x, y)	
x<1999 → y≤300	
∀	
----------	o

Figure 2 i-schema format and the item '*part/cost-price*'

Items and their Constraints

Items are a formalism for describing the things in an application and have three important properties: items have a uniform format no matter whether they represent data, information or knowledge things; items incorporate two distinct classes of constraints, and a single rule of "normalisation" can be specified for items (Debenham 1996c). The "meaning" of an item is called its *semantics* (Debenham 1996b). Items may be presented informally as "i-schema" or formally as λ-calculus expressions. Formally an item is a named triple $A[S_A, V_A, C_A]$ where A is the *item name*, S_A is called the *semantics* of *A*, V_A is called the *value constraints* of *A* and C_A is called the *set constraints* of *A*. The *i-schema* notation for representing items informally is shown in Figure 2. The λ-calculus is used to establish the formal properties of items. The i-schema are used to specify items in practice.

For example, an application could contain an association whereby each *part* is associated with a *cost-price*. This association could be subject to the value constraint that parts whose part-number is less that 1,999 will be associated with a cost price of no more than $300. This association could be subject to the universal set constraint that every part must be in this association, and the candidate set constraint that each part is associated with a unique cost-price. This association could be represented by the information item named *part/cost-price*. The i-schema for this information item is shown in Figure 2. The λ-calculus form for this item is:

$$\begin{aligned}
&\textit{part/cost-price}[\ \lambda xy\bullet[S_{part}(x) \wedge S_{cost\text{-}price}(y) \\
&\quad \wedge \text{costs}(x, y)]\bullet, \\
&\ \lambda xy\bullet[\ V_{part}(x) \wedge V_{cost\text{-}price}(y) \\
&\quad \wedge ((x < 1999) \rightarrow (y \leq 300))\ ]\bullet, \\
&\ C_{part} \wedge C_{cost\text{-}price} \wedge (\text{Uni}(\textit{part}) \\
&\quad \wedge \text{Can}(\textit{cost-price}, \{\textit{part}\}))_{\textit{part/cost-price}}\ ]
\end{aligned}$$

Rules, or knowledge, can also be defined as items and thus can be represented using i-schema. Consider the rule "the sale price of parts is the cost price marked up by a universal mark-up factor"; suppose that this rule is represented by the item named *[part/sale-price, part/cost-price, mark-up]*. The idea of defining the semantics of items as recognising functions for the members of their value set extends to complex, recursive knowledge items too (Debenham 1996b). For example, the semantics of the *[part/sale-price, part/cost-price, mark-up]* item is:

$$\begin{aligned}
&\lambda x_1 x_2 y_1 y_2 z\bullet[\ S_{part/sale\text{-}price}(x_1, x_2) \\
&\quad \wedge S_{part/cost\text{-}price}(y_1, y_2) \wedge S_{mark\text{-}up}(z) \\
&\quad \wedge ((x_1 = y_1) \rightarrow (x_2 = z \times y_2))]\bullet
\end{aligned}$$

Each item contains two types of constraints; these two types are the "value constraints" and the "set constraints". For example, consider the item *[part/sale-price, part/cost-price, mark-up]*, this item incorporates the value constraint that if this item is used to calculate the sale-price then the sale-price must be greater than the cost-price. This item also incorporates the set constraints:

$$\begin{aligned}
&\text{Uni}(\textit{part/sale-price}) \wedge \text{Uni}(\textit{part/cost-price}) \\
&\quad \wedge \text{Can}(\textit{part/sale-price}, \{\textit{part/cost-price}, \textit{mark-up}\}) \\
&\quad \wedge \text{Can}(\textit{part/cost-price}, \{\textit{part/sale-price}, \textit{mark-up}\}) \\
&\quad \wedge \text{Can}(\textit{mark-up}, \{\textit{part/sale-price}, \textit{part/cost-price}\})
\end{aligned}$$

where "Uni(*part/sale-price*)" requires that this item's value set must contain all values of the value set of the item "*part/sale-price*". Also "Can(*part/sale-price*, {*part/cost-price*, *mark-up*})" requires that in this item's value set the value set of the two items {*part/cost-price*, *mark-up*} functionally determine the value set of the item *part/sale-price*. The i-schema for this knowledge item shown in Figure 3.

Constraints which constrain the way that an item *is* are called *static constraints*. The examples given above are all static constraints. In contrast, *dynamic* item constraints are

[part/sale-price, part/cost-price, mark-up]		
part/sale-price	*part/cost-price*	*mark-up*
(x, w)	(x, y)	z
→ (w = z × y)		
→ w > y		
∀	∀	
----------		o
o	----------	
----------	o	----------

Figure 3 *[part/sale-price, part/cost-price, mark-up]*

name		object name
type1	type2	argument type
x	y	dummy variables
meaning of object		object semantics
constraints on values		object value constraints
set constraints		object set constraints

costs	
D^1	D^1
x	y
costs(x, y)	
x<1999 → y≤300	
∀	
----------	o

Figure 4 o-schema format and the object '*costs*'

mark-up-rule		
I^2	I^2	D^1
(x, w)	(x, y)	(z)
→ (w = z × y)		
→ w > y		
∀	∀	
----------		o
o	----------	
----------	o	----------

Figure 5 o-schema for object '*mark-up-rule*'

constraints on how an item may be modified. An example of a dynamic item constraint on the knowledge item above is the requirement that if it is modified for any reason then the values in its value set associated with the component "*part/sale-price*" can only change by less than 10% of the previous values.

A single rule of "normalisation" may be specified for items (Debenham 1995). This rule may be applied to complex items, including complex knowledge items, to break them down into simpler items. From this single rule the five normal forms for relational database (Date 1986) may be derived as well as a comprehensive set of new normal forms for knowledge (Debenham 1996c).

Objects and their Constraints

Knowledge items contain embedded within them the full specification of their constituent components. Knowledge items are incapable of representing the essence of rules. Objects are item building operators which have four important properties: objects have a uniform format no matter whether they represent data, information or knowledge things; objects two incorporate distinct classes of constraints; objects enable items to be built in such a way as to inherit the characteristics of their components, and a single rule of "normalisation" can be specified for objects such that a conceptual model built with "normal" object operators will contain only "normal" items.

Each object is an operator which turns n items into another item for some value of n. Further, the definition of each object will presume that the set of items to which that object may be applied are of a specific "type". The *type* of an m-adic item is determined both by whether it is a data item, an information item or a knowledge item and by the value of m. The *argument type* of an n-adic object is an n-tuple which specifies the types of the n items to which that object may be applied. Each of the n elements in an argument type will be "free" or "fixed". A *free argument type* is denoted by X^n and indicates that the object may be applied to any type of n-adic item and thus simply specifies the arity of that item. For example, if an object has argument type (X^2, X^2, X^1) then it may be applied to any 2-adic item, followed by any other 2-adic item which is followed by any 1-adic item. A *fixed argument type* is denoted by D^n (standing for "data"), I^n (standing for "information") or K^n (standing for "knowledge") and indicates that the object can only be applied to an n-adic item of the nominated type. A fixed argument type specifies both the arity of each argument and whether that argument should be a data item, an information item or a knowledge item.

The formal definition of an object is similar to that of an item. An object is a named, typed triple ***A***[E,F,G], where:

- ***A*** is a unique *object name* that, by convention, is written in bold italic script;
- E is an n-argument typed expression called the *object semantics*;
- F is an n-argument typed expression called the *object value constraints* and
- G is an n-argument typed expression called the *object set constraints*.

The object *value constraints* is an n-argument typed expression which is satisfied by the labels which belong to the value set of any item constructed by applying that object to a set of items. The object *set constraints* is an n-argument typed expression constructed from the primitives "Card", "Uni" and "Can" which represents structural constraints on that object and will be illustrated in the examples which follow.

As for items, objects may be presented informally as "o-schema" or formally as λ-calculus expressions. The *o-schema* notation for representing objects informally is shown in Figure 4.

The ***costs*** object of argument type (D^1, D^1) may be used to build the *part/cost-price* item:

costs(*part, cost-price*) = *part/cost-price*

The o-schema for the ***costs*** object is shown in Figure 4.

The ***mark-up-rule*** object of argument type (I^2, I^2, D^1) may be used to build the *[part/sale-price, part/cost-price, mark-up]* item:

[part/sale-price, part/cost-price, mark-up] =
mark-up-rule(*part/sale-price,*
part/cost-price, mark-up) =
mark-up-rule(***sells-for***(*part*, ***sell***(*price*)),
costs(*part*, *cost*(*price*)), *mark-up*)

The o-schema for the ***mark-up-rule*** object is shown in Figure 5.

Model Constraints.

Model constraints are constraints which apply to the whole conceptual model. An example of an "information model constraint" is now described. Consider any three sets of tuples which could form the value sets of the three items "*part/sale-price*", "*part/cost-price*" and "*mark-up*". For example consider the tuples shown in Figure 1. These tuples should be consistent with the knowledge represented by the knowledge item *[part/sale-price, part/cost-price, mark-up]*. Further an inference engine can demonstrate automatically that these tuples are consistent with that knowledge item. The constraint that the item *[part/sale-price, part/cost-price, mark-up]* should remains consistent with the particular information model shown in Figure 1 is an example of an information model constraint. Hand-coded, simple but non-trivial information models can be used as powerful information model constraints on knowledge items. Information model constraints are especially appropriate in applications where the knowledge is subject to a high rate of change and the essential meaning of the information model is fairly stable.

Knowledge Maintenance

The "coupling map" supports the maintenance procedure; it is a representation of the "coupling relationships". "Coupling relationships" link two items in the conceptual model if they share some common structure or common meaning. A *coupling relationship* joins two items in the conceptual model if modification of one of these items could require that the other item should be checked for correctness if the integrity of the conceptual model is to be preserved. The efficiency of maintenance procedures is substantially concerned with reducing the number of coupling relationships in the conceptual model (Lehner et al 1993). One kind of coupling relationship may be removed by applying a generalised form of knowledge normalisation (Debenham 1996c). Another kind of coupling relationship may be removed by applying referential constraints to items. Referential constraints complicate the item to which they are applied and should thus only be applied to items of low volatility.

The *conceptual model* consists of both a representation of the things in the applications as "items", and a coupling

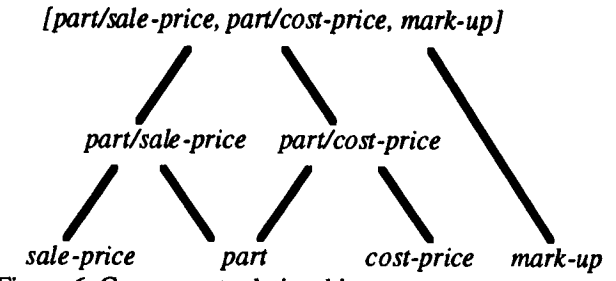

Figure 6 Component relationships

map. The coupling map is a representation of the coupling relationships. These coupling relationships are of four distinct kinds (Debenham 1996a). First, *duplicate relationships* link two items which share some common meaning. In other words, a duplicate relationship indicates that a real fact has been represented, at least in part, in more than one place. Second, *component relationships* link each item to its components. For example, the component relationships for the item *[part/sale-price, part/cost-price, mark-up]* above are shown in Figure 6. Third, *equivalence relationships* link two items whose semantics are logically equivalent. Fourth, *sub-item relationships* link two items one of whose semantics logically implies the other's semantics.

The coupling map can be simplified. The coupling map contains four kinds of coupling relationship. Duplicate relationships may be removed by applying the process of knowledge normalisation (Debenham 1996c). Some component relationships may be removed by applying referential constraints. Equivalence relationships may be removed by renaming. Sub-item relationships may be reduced to sub-type relationships.

Sub-item relationships join two items if one item is a sub-item of the other item. If a given data item is a sub-item of another data item then it is usual to say that the given data item is a *sub-type* of the other data item. For example, the item *part1* could have as its value set all valid spare part numbers which lie between 1 and 1999, and the item *part* could have as its value set all valid spare part numbers; in this example *part1* is a sub-type of *part*.

Sub-item relationships may exist between information items or knowledge items, but all sub-item relationships can be reduced to sub-type relationships between data items. For example consider the item *car-part/cost-price* shown in Figure 7. The structure at the top shows that this item is a sub-item of the item *part/cost-price*. The structure at the bottom shows how this sub-item relationship has been reduced to a sub-type relationship.

Normalisation

As in traditional database technology (Date 1986), the normalisation of the conceptual model is intended to produce a model that will support the maintenance process by preventing the introduction of update anomalies. Normalisation removes duplicate relationships from the coupling map.

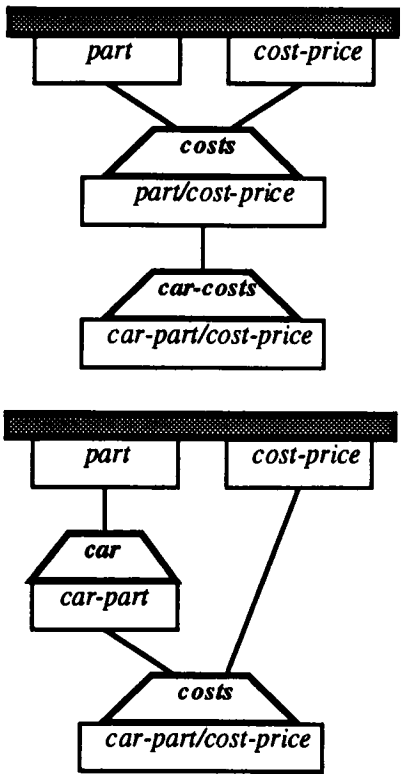

Figure 7 Reduction of sub-item relationship

In (Debenham 1996c) an "item join" operation is defined. Item join provides the basis for item decomposition. Given items A and B, the item with name $A \otimes_E B$ is called the *join* of A and B on E, where E is a set of components common to both A and B. When two items are joined on the component set which consists of *all* of their identical components we omit the subscript of the join operator. Using the rule of composition $\otimes$, knowledge items, information items and data items may be joined with one another regardless of type. For example, the knowledge item:

$$[cost\text{-}price,\ tax]\ [\lambda xy\bullet[x = y \times 0.05]\bullet,\ \lambda xy\bullet[x < y]\bullet,\ C_{[cost\text{-}price,\ tax]}]$$

can be joined with the information item *part/cost-price* on the set $\{cost\text{-}price\}$ to give the information item *part/cost-price/tax*. In other words:

$$[cost\text{-}price,\ tax] \otimes_{\{cost\text{-}price\}} part/cost\text{-}price = part/cost\text{-}price/tax[\lambda xyz\bullet[costs(x,y) \wedge z = y \times 0.05]\bullet,\ \lambda xyz\bullet[((1000<x<1999) \rightarrow (0<y\leq 300)) \wedge (z<y)]\bullet,\ C_{part/cost\text{-}price/tax}]$$

In this way items may be joined together to form more complex items. Alternatively, the $\otimes$ operator may form the basis of a theory of decomposition in which each item may be replaced by a set of simpler items. An item I is *decomposable* into the set of items $D = \{I_1, I_2, ..., I_n\}$ if:

- I_i has non-trivial semantics for all i,
- $I = I_1 \otimes I_2 \otimes ... \otimes I_n$, where
- each join is *monotonic*; that is, each term in this composition contributes at least one component to I.

If item I is decomposable then it will not necessarily have a unique decomposition. There is one rule of decomposition: "Given a conceptual model discard any items which are decomposable". This rule applies to all items. For example, this rule requires that the item *part/cost-price/tax* should be discarded in favour of the items *[cost-price, tax]* and *part/cost-price*.

A conceptual model is said to be *normal* if the items and objects in it are not decomposable. Suppose that item I has the three components A, B and C. Consider the different ways in which this item $I = I(C, B, A)$ can be decomposed into sub-items I_1, I_2 and I_3. These different ways are categorised by the different ways in which functional associations are present in I; functional associations are represented by the candidate constraint. The different ways in which functional associations are present in a three component item lead precisely to the classical normal forms:

3NF $I((C, B) \Leftarrow A) = I_2(C \Leftarrow B) \otimes_{\{B\}} I_1(B \Leftarrow A)$

2NF $I(C \Leftarrow (B, A)) = I_2(C \Leftarrow B) \otimes_{\{B\}} I_1(B, A)$

BC $I(B \Leftarrow (C, A)) = I_2(C \Leftarrow B) \otimes_{\{B\}} I_1(B, A)$

4NF $I(C, B, A) = I_2(C, B) \otimes_{\{B\}} I_1(B, A)$

5NF $I(C, B, A) = I_3(A, C) \otimes_{\{C\}} I_2(C, B) \otimes_{\{B\}} I_1(B, A)$

The classical normal forms noted above apply equally well to knowledge as to data or information (Debenham 1996b). Thus the classical normal forms provide a complete characterisation of the different ways in which an item of three components may be decomposed. Further normal forms may be derived by considering decompositions of items of more than three components.

Referential Constraints

Once the internal model has been derived, referential constraints may be applied to the items in the conceptual model. *Referential constraints* state that a particular component relationship need not be followed during the complete execution of a maintenance operation. These referential constraints prune the component relationships and thus simplify the coupling map. Referential constraints improve the efficiency of the maintenance procedure.

The maintenance procedure is guided by the coupling map. This procedure is activated by the modification of an item. An item's semantics recognises the members of its value set. Thus if an item's value set is modified then that item's semantics has been modified. For example, the value set of the *part/cost-price* item may be stored as a relation R. The predicate "costs(x,y)" occurs in the semantics of this item. In strict terms, the meaning of this predicate is "x costs y at time τ". If a tuple is added, modified or deleted in the relation R then all of the coupling relationships from the item *part/cost-price* must, in theory, be investigated. In other words, each simple

maintenance task on a relation can generate a significant maintenance task. Referential constraints may be applied to isolate the effect of simple maintenance tasks.

For example, consider a simple example in which *[part/sale-price, part/cost-price, mark-up]* is the only knowledge item in the conceptual model. Suppose that the internal model states that the *mark-up* data item and the *part/cost-price* information item should both be physically stored. That is, of the three distinct if-then interpretations of this single knowledge item, only the interpretation which derives the value set of the *part/sale-price* information item is required. The constraint that "the value set of the information item *part/cost-price* is *fixed in* the knowledge item *[part/sale-price, part/cost-price, mark-up]*" is an example of a referential constraint. This constraint means that if the tuples in the relation part/cost-price are modified then it is not necessary to follow the component link to the item *[part/sale-price, part/cost-price, mark-up]*. In other words "the validity of the knowledge item *[part/sale-price, part/cost-price, mark-up]* is invariant of the contents of the value set of its component information item *part/cost-price*". This referential constraint is a static constraint on the item *[part/sale-price, part/cost-price, mark-up]*; it states that this knowledge item must apply to any tuple which satisfies the item constraints of the information item *part/cost-price*. This constraint prunes the component relationship *from* the information item *part/cost-price to* the knowledge item *[part/sale-price, part/cost-price, mark-up]* in the instance when the value set of item *part/cost-price* is modified.

The referential constraint just considered has the effect of pruning a component relationship *from* an information item *to* a knowledge item. Component relationships *from* knowledge items *to* their constituent data or information items can be pruned in a similar way. For example, the constraint that "the value set of the knowledge item *[part/sale-price, part/cost-price, mark-up]*" is *fixed on* the information item *part/cost-price* " is an example of a referential constraint. This constraint means that if the item *[part/sale-price, part/cost-price, mark-up]* is modified then it is not necessary to follow the component link to the tuples of the relation part/cost-price. In other words "the validity of the value set of the information item *part/cost-price* is invariant of the item *[part/sale-price, part/cost-price, mark-up]*". This referential constraint is a static constraint on the information item *part//cost-price*. This constraint prunes the component relationship *from* the knowledge item *[part/sale-price, part/cost-price, mark-up] to* the information item *part/cost-price* in the instance when the clauses which implement the item *[part/sale-price, part/cost-price, mark-up]* are modified.

Summary

Constraints are a management tool for knowledge-based systems. Pragmatic constraints may be applied to protect knowledge-based systems against the introduction of update anomalies during the execution of maintenance operations. A taxonomy of pragmatic knowledge constraints has been proposed. Referential knowledge constraints contribute to the efficiency of the knowledge-based systems maintenance process. The efficiency of the maintenance procedure is increased further if the knowledge has been normalised.

References

Coenen, F.; and Bench-Capon, T. 1992. Building Knowledge Based Systems for Maintainability. In proceedings Third International Conference on Database and Expert Systems Applications DEXA'92, Valencia, Spain, September, 1992, pp415-420.

Date, C.J. 1986. *An Introduction to Database Systems.* (4th edition) Addison-Wesley, 1986.

Debenham, J.K. 1989. *Knowledge Systems Design.* Prentice Hall, 1989.

Debenham, J.K. 1995. Understanding Expert Systems Maintenance. In proceedings Sixth International Conference on Database and Expert Systems Applications DEXA'95, London, September 1995.

Debenham, J.K. 1996a. Characterising Maintenance Links. In proceedings Third World Congress on Expert Systems, Seoul, February 1996.

Debenham, J.K. 1996b. Integrating Knowledge Base and Database. In proceedings 10th ACM Annual Symposium on Applied Computing SAC'96, Philadelphia, February 1996, pp28-32.

Debenham, J.K. 1996c. Knowledge Simplification. In proceedings 9th International Symposium on Methodologies for Intelligent Systems ISMIS'96, Zakopane, Poland, June 1996.

Debenham, J.K. 1996d. Unification of Knowledge Acquisition and Knowledge Representation. In proceedings International Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems IPMU'96, Granada, Spain, July 1996.

Debenham, J.K.; and Devedžić, V. 1996a. Designing Knowledge-Based Systems for Optimal Performance. In proceedings Seventh International Conference on Database and Expert Systems Applications DEXA'96, Zurich, Switzerland, September 1996, pp728-737.

Debenham, J.K.; and Devedžić, V. 1996b. Knowledge Analysis in KBS Design. In proceedings Seventh International Conference on Artificial Intelligence: Methodologies, Systems, Applications AIMSA'96, Sozopol, Bulgaria, September 1996, pp178-190.

Devedžić V. 1996. Organization and Management of Knowledge Bases: An Object-Oriented Approach. In proceedings of The Third World Congress on Expert Systems, Vol.II, Seoul, Korea, 1996, pp. 1263-1270.

Gray, P.M.D. 1989. Expert Systems and Object-Oriented Databases: Evolving a New Software Architecture. In

Research and Development in Expert Systems V, Cambridge University Press, 1989, pp 284-295.

Kang, B.; Gambetta, W.; and Compton, P. 1996. Validation and Verification with Ripple Down Rules. *International Journal of Human Computer Studies* Vol 44 (2) pp257-270 (1996).

Lehner, F.; Hofman, H.F.; Setzer, R.; and Maier, R. 1993. Maintenance of Knowledge Bases. In proceedings Fourth International Conference DEXA93, Prague, September 1993, pp436-447.

Tayar, N. 1993. A Model for Developing Large Shared Knowledge Bases. In proceedings Second International Conference on Information and Knowledge Management, Washington, November 1993, pp717-719.