

A Lesson for Software Engineering from Knowledge Engineering

John Debenham

University of Technology, Sydney,
Faculty of Information Technology,
PO Box 123, NSW 2007, Australia
debenham@it.uts.edu.au

Abstract. Knowledge engineering has developed fine tools for maintaining the integrity of knowledge bases. These tools may be applied to the maintenance of conventional programs particularly those programs in which business rules are embedded. A unified model of knowledge represents business rules at a higher level of abstraction than the rule-based paradigm. Representation at this high level of abstraction enables any changes to business rules to be quantified and tracked through to the imperative programs that implement them. Further, methods may be applied to simplify the unified model so that the maintenance of the imperative implementation too is simplified.

1 Introduction

A very simple example motivates this discussion. Consider the business rule: [K1] “The sale price of a part is the cost price of that part marked up by the markup rate for that part”. This rule is not in if-then form. Its form is that of a simple statement of fact. It could give rise to the following simple Java program:

```
public int part_sale_price( int part_no, int part_cost_markup[] [],
                           int no_of_part_nos ) {
    if (no_of_part_nos < 1 ) return -1;
    for ( int count = 0; count < no_of_part_nos; count++ ) {
        if ( part_cost_markup[count][0] == part_no ) {
            return part_cost_markup[count][1] *
                   part_cost_markup[count][2] / 100;
        }
    }
    return -2;
}
[P1]
```

that returns the sale_price of a given part_no. There is no immediate issue here. But if the wisdom in [K1] finds its way into other imperative representations then some machinery is required to acknowledge that those representations are linked. Further, as we will see, [K1] may contain within it other business rules that may also be buried in yet others. The argument presented here is that these links are revealed by applying knowledge engineering tools to the construction of a conceptual model

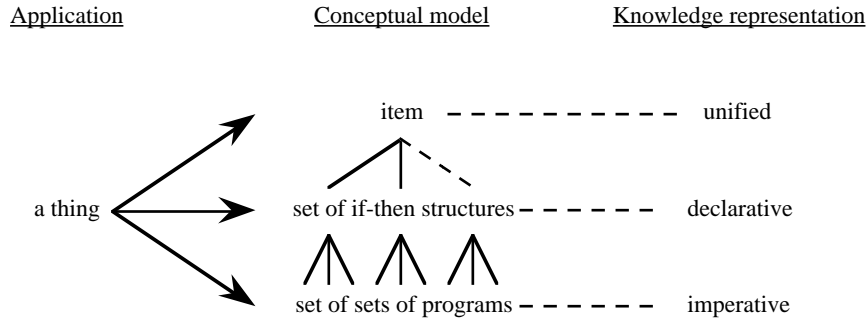


Fig. 1. A thing and its representation in the unified representation, a declarative representation and an imperative interpretation.

that is employed as a specification of the imperative implementations derived from it. Further, methods may be applied to convert the conceptual model into a model in which the original implicit links become explicit. This exposes the maintenance structure so that it may then form an integral part of either a declarative or an imperative implementation.

To illustrate the complexity present in this very simple example, consider also the business rules: [K2] “The profit on a part is the difference between the marked-up cost price and the cost price”, and [K3] “The profit on a part is the difference between the sale price and the cost price of that part”. [K2] may be derived from both [K1] and [K3]. Why does this present a problem? If the knowledge in either [K1] or [K3] becomes invalid and so is modified then [K2] should be modified as well to preserve consistency. The relationship between these three chunks is not difficult to identify given the raw chunks, but, given only implementations as programs—even with documentation—the relationship becomes more obscure.

An abstract conceptual model that describes the knowledge embedded in a set of programs is used to drive the maintenance process for those programs. The maintenance problem is to determine which programs should be checked for correctness in response a change in the application [1]. This model is at a very high level of abstraction. In it each chunk of knowledge is represented directly as a single ‘item’. Each item may be interpreted as a number of declarative if-then rules. Each rule may be interpreted as a number of imperative programs. So in the model an item will correspond to a number of programs each of which implements an imperative interpretation of the original chunk of knowledge. This abstraction hierarchy is represented in Fig. 1.

In the above example, business rule [K1] admits three different declarative if-then interpretations:

$$\begin{aligned} \text{part/sale-price}(x, y) \leftarrow & \text{part/cost-price}(x, z), \\ & \text{part/mark-up}(x, w), y = (z \times w) \end{aligned} \quad \text{[C1.1]}$$

$$\begin{aligned} \text{part/cost-price}(x, z) \leftarrow & \text{part/sale-price}(x, y), \\ & \text{part/mark-up}(x, w), y = (z \times w) \end{aligned} \quad \text{[C1.2]}$$

$$\begin{aligned} \text{part/mark-up}(x, w) \leftarrow & \text{part/sale-price}(x, y), \\ & \text{part/cost-price}(x, z), y = (z \times w) \end{aligned} \quad \text{[C1.3]}$$

<i>part/sale-price</i>		<i>part/cost-price</i>		<i>part/mark-up</i>	
1234	1.44	1234	1.20	1234	1.2
2468	2.99	2468	2.30	2468	1.3
3579	4.14	3579	3.45	3579	1.2
1357	10.35	1357	4.50	1357	2.3
9753	12.06	9753	6.70	9753	1.8
8642	12.78	8642	8.52	8642	1.5
4321	5.67	4321	2.70	4321	2.1

Fig. 2. Value set for a knowledge item.

and each of these admit a number of imperative interpretations. Program [P1] is but one interpretation of [C1.1]. Note that [C1.1] as a Prolog program can find a *part_number* with a given cost—a task that [P1] can not do directly.

Given any form of conceptual model *maintenance links* are introduced that join two things in that model if a modification to one of them means that the other must necessarily be checked for correctness, and so possibly modified, if consistency of that model is to be preserved. If that other thing requires modification then the links from it to yet other things are followed, and so on until things are reached that do not require modification. If node A is linked to node B which is linked to node C then nodes A and C are *indirectly* linked. In a *coherent* model of an application everything is indirectly linked to everything else. A good conceptual model for maintenance will have a low density of maintenance links [2]. Ideally, the set of maintenance links will be *minimal* in that none may be removed. Informally, one conceptual model is “better” than another if it leads to less checking for correctness. The aim of this work is to generate a good conceptual model. A classification of maintenance links into four classes is given here. Methods are given for removing two of these classes of link so reducing the density of maintenance links in the resulting model. In this way the maintenance problem is simplified.

Approaches to the maintenance of declarative conceptual models are principally of two types [3]. First, approaches that take a model ‘as is’ and then try to *control* the maintenance process [4]. Second, approaches that *engineer* a model so that it is in a form that is inherently easy to maintain [5] [6]. The approach described here is of the second type because maintenance is driven by a maintenance link structure that is simplified by transforming the model.

2. Representing Knowledge

Consider again the chunk of knowledge [K1]. Suppose that that chunk is represented as an *item*—with the name [*part/sale-price, part/cost-price, part/mark-up*]. The data associated with this item may be presented as a rather messy relation; such a relation is called the *value set* of the item, a possible value set is shown in Fig. 2.

The meaning of an item *A*—called its *semantics* S_A —is an expression that recognises the members of that item’s value set. For the item considered above the semantics could be:

$$\lambda x_1 x_2 y_1 y_2 z_1 z_2 \bullet [(S_{part/sale-price}(x_1, x_2) \wedge S_{part/cost-price}(y_1, y_2) \wedge S_{part/mark-up}(z_1, z_2)) \wedge ((x_1 = y_1 = z_1) \rightarrow (x_2 = z_2 \times y_2))] \bullet$$

where $S_{part/sale-price} = \lambda xy \bullet [S_{part}(x) \wedge S_{sale-price}(y) \wedge \text{sells-for}(x, y)] \bullet$
and where $S_{part} = \lambda x \bullet [\text{is-a}[x:P]] \bullet$ for some suitable domain P where:

$$\text{is-a}[x:P] \begin{cases} = \mathbf{T} & \text{if } x \text{ is in } P \\ = \mathbf{F} & \text{otherwise} \end{cases}$$

In general, an *item* is a named triple $A[S_A, V_A, C_A]$ with *item name* A , S_A is called the *item semantics* of A , V_A is called the *item value constraints* of A and C_A is called the *item set constraints* of A . The item semantics, S_A , is a λ -calculus expression that recognises the members of the value set of item A . The expression for an item's semantics may contain the semantics of other items $\{A_1, \dots, A_n\}$ called that item's *components*:

$$\lambda y_1^1 \dots y_{m_1}^1 \dots y_{m_n}^n \bullet [S_{A_1}(y_1^1, \dots, y_{m_1}^1) \wedge \dots \wedge S_{A_n}(y_1^n, \dots, y_{m_n}^n) \wedge J(y_1^1, \dots, y_{m_1}^1, \dots, y_{m_n}^n)] \bullet$$

The item value constraints, V_A , is a λ -calculus expression:

$$\lambda y_1^1 \dots y_{m_1}^1 \dots y_{m_n}^n \bullet [V_{A_1}(y_1^1, \dots, y_{m_1}^1) \wedge \dots \wedge V_{A_n}(y_1^n, \dots, y_{m_n}^n) \wedge K(y_1^1, \dots, y_{m_1}^1, \dots, y_{m_n}^n)] \bullet$$

that should be satisfied by the members of the value set of item A as they change in time; so if a tuple satisfies S_A then it should satisfy V_A [7]. The expression for an item's value constraints contains the value constraints of that item's *components*. The item set constraints, C_A , is an expression of the form:

$$C_{A_1} \wedge C_{A_2} \wedge \dots \wedge C_{A_n} \wedge (L)_A$$

where L is a logical combination of:

- Card lies in some numerical range;
- $\text{Uni}(A_i)$ for some i , $1 \leq i \leq n$, and
- $\text{Can}(A_i, X)$ for some i , $1 \leq i \leq n$, where X is a non-empty subset of $\{A_1, \dots, A_n\} - \{A_i\}$;

subscripted with the name of the item A , “ $\text{Uni}(a)$ ” means that “all members of the value set of item a must be in this association”. “ $\text{Can}(b, A)$ ” means that “the value set of the set of items A functionally determines the value set of item b ”. “Card” means “the number of things in the value set”. The subscripts indicate the item's components to which that set constraint applies.

For example, each *part* may be associated with a *cost-price* subject to the “value constraint” that parts whose part-number is less than 1,999 should be associated with a cost price of no more than \$300. A set constraint specifies that every part must be in this association, and that each part is associated with a unique cost-price. The information item named *part/cost-price* then is:

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

Rules, or knowledge, can also be defined as items, although it is neater to define knowledge items using “objects”. “Objects” are item building operators. The knowledge item [K1] $[part/sale-price, part/cost-price, part/mark-up]$ which means “The sale price of a part is the cost price of that part marked up by the markup rate for that part” is:

$$\begin{aligned}
& [part/sale-price, part/cost-price, part/mark-up][\\
& \lambda x_1 x_2 y_1 y_2 z_1 z_2 \bullet [(S_{part/sale-price}(x_1, x_2) \wedge S_{part/cost-price}(y_1, y_2) \wedge \\
& S_{part/mark-up}(z_1, z_2)) \wedge ((x_1 = y_1 = z_1) \rightarrow (x_2 = z_2 \times y_2))] \bullet, \\
& \lambda x_1 x_2 y_1 y_2 z_1 z_2 \bullet [V_{part/sale-price}(x_1, x_2) \wedge V_{part/cost-price}(y_1, y_2) \wedge \\
& V_{part/mark-up}(z_1, z_2)) \wedge ((x_1 = y_1) \rightarrow (x_2 > y_2))] \bullet, \\
& C_{[part/sale-price, part/cost-price, mark-up]}]
\end{aligned}$$

What have we achieved with the representation of our business rule [K1]? All of what it says plus additional constraints is represented above as an item. The formal notation is not particularly “user friendly”. An alternative schema notation is more palatable for practical use [1]. But the item above is a complete formal representation of the business rule. Further we will show that items may be modified so as to simplify the maintenance links in the conceptual model and so too the links from the model to an imperative program implementation of it.

Two different items can share common knowledge and so can lead to a profusion of maintenance links. This problem can be avoided by using objects. An n-adic *object* is an operator that maps n given 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 type is denoted respectively by \mathbf{D}^m , \mathbf{I}^m and \mathbf{K}^m . Items may also have unspecified, or free, type which is denoted by \mathbf{X}^m . The formal definition of an object is similar to that of an item. An *object* named A is a typed triple $A[E,F,G]$ where E is a typed expression called the *semantics* of A, F is a typed expression called the *value constraints* of A and G is a typed expression called the *set constraints* of A. For example, the *part/cost-price* item can be built from the items *part* and *cost-price* using the *costs* operator:

$$\begin{aligned}
& part/cost-price = costs(part, cost-price) \\
& costs[\lambda P:\mathbf{X}^1 Q:\mathbf{X}^1 \bullet \lambda xy \bullet [S_P(x) \wedge S_Q(y) \wedge costs(x,y)] \bullet \bullet, \\
& \lambda P:\mathbf{X}^1 Q:\mathbf{X}^1 \bullet \lambda xy \bullet [V_P(x) \wedge V_Q(y) \wedge ((1000 < x < 1999) \rightarrow (y \leq 300))] \bullet \bullet, \\
& \lambda P:\mathbf{X}^1 Q:\mathbf{X}^1 \bullet [C_P \wedge C_Q \wedge (Uni(P) \wedge Can(Q, \{P\}))] \bullet]
\end{aligned}$$

where $\mathcal{V}(costs, P, Q)$ is the name of the item $costs(P, Q)$.

Data objects provide a representation of sub-typing. Rules are quite clumsy when represented as items; objects provide a far more compact representation. For example, consider the $[part/sale-price, part/cost-price, part/mark-up]$ knowledge item which represents the rule “parts are marked-up by a universal mark-up factor”. This item can

be built by applying a knowledge object *mark-up-rule* of argument type $(\mathbf{I}^2, \mathbf{I}^2, \mathbf{I}^2)$ to the items *part/sale-price*, *part/cost-price* and *part/mark-up*. That is:

$$[part/sale-price, part/cost-price, mark-up] = \mathbf{mark-up-rule}(part/sale-price, part/cost-price, part/mark-up)$$

Objects also represent value constraints and set constraints in a uniform way. A decomposition operation for objects is defined in [1].

A *conceptual model* consists of a set of items and a set of maintenance links. The items are constructed by applying a set of object operators to a set of fundamental items called the *basis*. The *maintenance links* join two items if modification to one of them necessarily means that the other item has at least to be checked for correctness if consistency is to be preserved. Item join provides the basis for item decomposition [8]. 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*. 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:

$$[cost-price, tax] [\lambda xy \bullet [S_{cost-price}(x) \wedge S_{tax}(y) \wedge x = y \times 0.05] \bullet, \\ \lambda xy \bullet [V_{cost-price}(x) \wedge V_{tax}(y) \wedge x < y] \bullet, \\ C_{[cost-price, tax]}]$$

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

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

In this way items may be joined together to form more complex items. The \otimes operator also forms the basis of a theory of decomposition in which each item is replaced by a set of simpler items. An item *I* is *decomposable* into the set of items $D = \{I_1, I_2, \dots, I_n\}$ if: I_i has non-trivial semantics for all *i*, $I = I_1 \otimes I_2 \otimes \dots \otimes I_n$, 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. The \otimes operator is applied to objects in a similar way [2]. The rule of decomposition is: "Given a conceptual model discard any items and objects which are decomposable". For example, this rule requires that the item *part/cost-price/tax* should be discarded in favour of the two items *[cost-price, tax]* and *part/cost-price*.

3. Maintenance Links

So far we have shown how business rules such as [K1] may be represented, manipulated and decomposed. We now address the maintenance of the conceptual model. A *maintenance link* joins two items in the conceptual model if modification of one item means that the other item must be checked for correctness, and maybe modified, if the consistency of the conceptual model is to be preserved [9]. The number of maintenance links can be very large. So maintenance links can only form the basis of a practical approach to knowledge base maintenance if there is some way of reducing their density on the conceptual model [10].

For example, given two items A and B , where both are n -adic items with semantics S_A and S_B respectively, if π is permutation such that:

$$(\forall x_1 x_2 \dots x_n) [S_A(x_1, x_2, \dots, x_n) \leftarrow S_B(\pi(x_1, x_2, \dots, x_n))]$$

then item B is a *sub-item* of item A . These two items should be joined with a maintenance link. If A and B are both data items then B is a *sub-type* of A . Suppose that:

$$X = E D; \text{ where } D = C A B \quad (1)$$

for items X, D, A and B and objects E and C . Item X is a sub-item of item D . Object E has the effect of extracting a sub-set of the value set of item D to form the value set of item X . Item D is formed from items A and B using object C . Introduce two new objects F and J . Suppose that object F when applied to item A extracts the same subset of item A 's value set as E extracted from the "left-side" (ie. the "A-side") of D . Likewise J extracts the same subset of B 's value set as E extracted from D . Then:

$$X = C G K; \text{ where } G = F A \text{ and } K = J B \quad (2)$$

so G is a sub-item of A , and K is a sub-item of B . The form (2) differs from (1) in that the sub-item maintenance links have been moved one layer closer to the data item layer, and object C has moved one layer away from the data item layer. Using this method repeatedly sub-item maintenance links between non-data items are reduced to sub-type links between data items.

It is shown now that there are four kinds of maintenance link in a conceptual model built using the unified knowledge representation. Consider two items A and B , and suppose that their semantics S_A and S_B have the form:

$$\begin{aligned} S_A &= \lambda y_1^1 \dots y_{m_1}^1 \dots y_{m_p}^p \bullet [S_{A_j}(y_1^1, \dots, y_{m_1}^1) \wedge \dots \wedge \\ &\quad S_{A_p}(y_1^p, \dots, y_{m_p}^p) \wedge J(y_1^1, \dots, y_{m_1}^1, \dots, y_{m_p}^p)] \bullet \\ S_B &= \lambda y_1^1 \dots y_{n_1}^1 \dots y_{n_q}^q \bullet [S_{B_j}(y_1^1, \dots, y_{n_1}^1) \wedge \dots \wedge \\ &\quad S_{B_q}(y_1^q, \dots, y_{n_q}^q) \wedge K(y_1^1, \dots, y_{n_1}^1, \dots, y_{n_q}^q)] \bullet \end{aligned}$$

S_A contains $(p + 1)$ terms and S_B contains $(q + 1)$ terms. Let μ be a maximal sub-expression of $S_A \otimes B$ such that:

$$\text{both } S_A \rightarrow \mu \text{ and } S_B \rightarrow \mu \quad (a)$$

where μ has the form:

$$\lambda y_1^1 \dots y_{d_1}^1 \dots y_{d_r}^r \cdot [S_{C_l}(y_1^1, \dots, y_{d_1}^1) \wedge \dots \wedge S_{C_r}(y_1^r, \dots, y_{d_r}^r) \wedge L(y_1^1, \dots, y_{d_1}^1, \dots, y_{d_r}^r)] \cdot$$

If μ is empty, ie. 'false', then the semantics of A and B are independent. If μ is non-empty then the semantics of A and B have something in common and A and B should be joined with a maintenance link.

Now examine μ to see *why* A and B should be joined. If μ is non-empty and if both A and B are items in the basis then:

A and B are a pair of basis items with logically dependent semantics (b)

If μ is non-empty and if A is *not* in the basis then there are three cases. First, if:

$$S_A \Leftrightarrow S_B \Leftrightarrow \mu \quad (c)$$

then items A and B are equivalent and should be joined with an *equivalence link*.

Second if (c) does not hold and:

$$\text{either } S_A \Leftrightarrow \mu \text{ or } S_B \Leftrightarrow \mu \quad (d)$$

then either A is a sub-item of B , or B is a sub-item of A and these two items should be joined with a *sub-item link*. Third, if (c) and (d) do not hold then if Δ is a minimal sub-expression of S_A such that $\Delta \rightarrow \mu$. Then:

$$\text{either } S_{A_j}(y_1^j, \dots, y_{m_j}^j) \in \Delta, \text{ for some } j \quad (e)$$

$$\text{or } J(y_1^1, \dots, y_{m_j}^1, \dots, y_{m_p}^p) \in \Delta \quad (f)$$

Both (e) and (f) may hold. If (e) holds then items A and B share one or more component items to which they should each be joined with a *component link*. If (f) holds then items A and B may be constructed with two object operators whose respective semantics are logically dependent. Suppose that item A was constructed by object operator C then the semantics of C will imply:

$$\begin{aligned} \Phi = & \lambda Q_1 : \mathbf{X}_1^{i_1} Q_2 : \mathbf{X}_2^{i_2} \dots Q_j : \mathbf{X}_j^{i_j} \cdot \lambda y_1^1 \dots y_{d_1}^1 \dots y_{d_r}^r \cdot [\\ & S_{P_1}(y_1^1, \dots, y_{d_1}^1) \wedge \dots \wedge S_{P_r}(y_1^r, \dots, y_{d_r}^r) \wedge \\ & L(y_1^1, \dots, y_{d_1}^1, \dots, y_{d_r}^r)] \cdot \end{aligned}$$

where the Q_i 's take care of any possible duplication in the P_j 's. Let E be the object $E[\Phi, \mathbf{T}, \emptyset]$ then C is a sub-object of E ; that is, there exists a non-tautological object F such that:

$$C \simeq_w E \otimes_M F \quad (g)$$

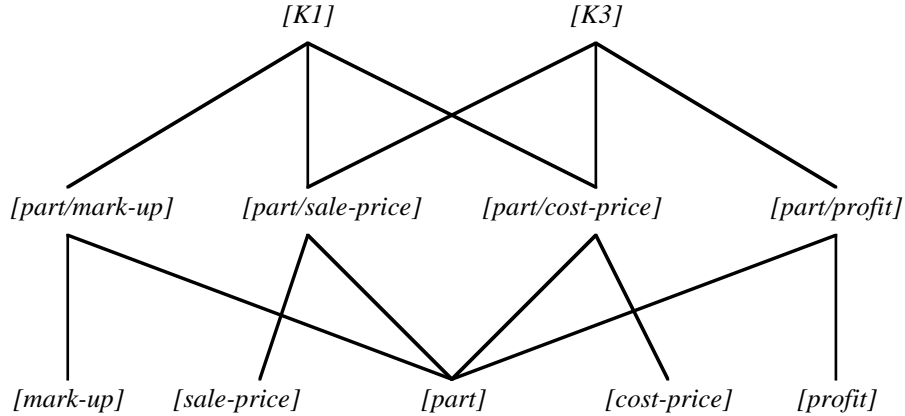


Fig. 3. Maintenance links for [K1] and [K2].

for some set M and where the join is not necessarily monotonic. Items A and B are *weakly equivalent*, written $A \simeq_w B$, if there exists a permutation π such that:

$$(\forall x_1 x_2 \dots x_n)[S_A(x_1, x_2, \dots, x_n) \Leftrightarrow S_B(\pi(x_1, x_2, \dots, x_n))]$$

where the x_i are the n_i variables associated with the i 'th component of A . If A is a sub-item of B and if B is a sub-item of A then items A and B are weakly equivalent.

If (g) holds then the maintenance links are of three different kinds. If the join in (g) is monotonic then (g) states that C may be decomposed into E and F . If the join in (g) is *not* monotonic then (g) states that either $C \simeq_w E$ or $C \simeq_w F$. So, if the join in (g) is not monotonic then *either* E will be weakly equivalent to C , *or* C will be a sub-object of E .

It has been shown above that sub-item links between non-data items may be reduced to sub-type links between data items. So if:

- the semantics of the items in the basis are all logically independent;
- all equivalent items and objects have been removed by re-naming, and
- sub-item links between non-data items have been reduced to sub-type links between data items

then the maintenance links will be between nodes marked with:

- a data item that is a sub-type of the data item marked on another node, these are called the *sub-type links*;
- an item and the nodes marked with that item's components, these are called the *component links*, and
- an item constructed by a decomposable object and nodes constructed with that object's decomposition, these are called the *duplicate links*.

If the objects employed to construct the conceptual model have been decomposed then the only maintenance links remaining will be the sub-type links and the component links. The sub-type links and the component links cannot be removed from the conceptual model.

4. Conclusion

A very high level conceptual model represents each chunk of knowledge as a single item. A rule of decomposition is applied to reduce [K2] above to [K1] and [K3], so removing [K2] from the conceptual model. Maintenance links join 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 validity of the conceptual model is to be preserved. The efficiency of maintenance procedures depends on a method for reducing the density of the maintenance links in the conceptual model. One kind of maintenance link is removed by applying the rule of knowledge decomposition [11]. Another is removed by reducing sub-item relationships to sub-type relationships [2]. And another is removed by re-naming. In the simple example given the conceptual model consists only of [K1] and [K3], and the maintenance links are just the component links as shown in Fig. 3. So what? Because the model can not be decomposed it means that the maintenance links in Fig. 3 are *complete*. These links may then be mapped to the imperative programs that implement the knowledge in chunks [K1] and [K2].

References

- [1] Debenham, J.K. "Knowledge Object Decomposition" in *proceedings 12th International FLAIRS Conference*, Florida, May 1999, pp203-207.
- [2] Mayol, E. and Teniente, E. "Addressing Efficiency Issues During the Process of Integrity Maintenance" in *proceedings Tenth International Conference DEXA99*, Florence, September 1999, pp270-281.
- [3] Katsuno, H. and Mendelzon, A.O. "On the Difference between Updating a Knowledge Base and Revising It", in *proceedings Second International Conference on Principles of Knowledge Representation and Reasoning, KR'91*, Morgan Kaufmann, 1991.
- [4] Barr, V. "Applying Reliability Engineering to Expert Systems" in *proceedings 12th International FLAIRS Conference*, Florida, May 1999, pp494-498.
- [5] Jantke, K.P. and Herrmann, J.. "Lattices of Knowledge in Intelligent Systems Validation" in *proceedings 12th International FLAIRS Conference*, Florida, May 1999, pp499-505.
- [6] Darwiche, A. "Compiling Knowledge into Decomposable Negation Normal Form" in *proceedings International Joint Conference on Artificial Intelligence, IJCAI'99*, Stockholm, Sweden, August 1999, pp 284-289.
- [7] Johnson, G. and Santos, E. "Generalizing Knowledge Representation Rules for Acquiring and Validating Uncertain Knowledge" in *proceedings 13th International FLAIRS Conference*, Florida, May 2000, pp186-2191.
- [8] Debenham, J.K. "*Knowledge Engineering*", Springer-Verlag, 1998.
- [9] Ramirez, J. and de Antonio, A. "Semantic Verification of Rule-Based Systems with Arithmetic Constraints" in *proceedings 11th International Conference DEXA2000*, London, September 2000, pp437-446.
- [10] Kern-Isberner, G. "Posulates for conditional belief revision" in *proceedings International Joint Conference on Artificial Intelligence, IJCAI'99*, Stockholm, Sweden, August 1999, pp 186-191.
- [11] Debenham, J.K. "From Conceptual Model to Internal Model", in *proceedings Tenth International Symposium on Methodologies for Intelligent Systems ISMIS'97*, Charlotte, October 1997, pp227-236.