

The Degradation of Knowledge Base Integrity

John Debenham

School of Computing Sciences
University of Technology, Sydney
PO Box 123 Broadway, NSW 2007, Australia
debenham@socs.uts.edu.au

Abstract

In a unified knowledge representation data, information and knowledge are all represented in a single formalism as “items”. Objects are item building operators. Items and objects contain two types of acceptability measures that measure the invalidity of item instances. A quantitative calculus estimates the extent to which knowledge base integrity may be expected to degrade as time passes. This calculus is simplified by the use of the unified knowledge representation. Expressions in this calculus are simplified if the knowledge has been decomposed.

Introduction

The terms ‘data’, ‘information’ and ‘knowledge’ are used here in a rather idiosyncratic sense [1]. The *data* in an application are those things that can be represented as simple constants or variables. The *information* is those things that can be represented as tuples or relations. The *knowledge* is those things that can be represented either as programs in an imperative language or as rules in a declarative language. A unified knowledge representation for conceptual modelling is described in [1]. That representation is *unified* in the sense that no distinction is made between the knowledge, information and data throughout the design process. A conceptual model is expressed in terms of “items” and “objects” [2]; objects are item-building operators. A single rule for “knowledge decomposition” [3] simplifies the maintenance of the conceptual model and its implementation. Classical database normalisation [4] is a special case of that single rule. The conceptual model reported in [1] is extended here to describe the degradation of knowledge integrity.

Approaches to the *preservation* of knowledge base integrity *either* aim to design the knowledge base so that it is inherently maintainable [1] *or* to present the knowledge in a form that discourages the introduction of inconsistencies [2]. Constraints are seldom mentioned in connection with knowledge; they can play a useful role in preserving knowledge base integrity. An approach to specifying knowledge constraints is described in [1]. Those constraints are two-valued in the sense that either they are satisfied or they aren’t. The *constraint domain* is the union of all knowledge base instances that satisfy a given set of knowledge constraints. The constraint domain

can be visualised as an area within which a given knowledge base should reside, and outside which the integrity of chunks of knowledge should be questioned.

A calculus estimates the degradation of knowledge integrity. The definition of this calculus is simplified by the unified knowledge representation. Expressions in this calculus are simplified if the knowledge has been decomposed. The work described here has a rigorous, formal theoretical basis expressed in terms of the λ -calculus; the work may also presented informally in terms of schema. Schema are used when the methodology is applied in practice.

Items and Objects

Items are a formalism for describing the things in an application; they have a uniform format no matter whether they represent data, information or knowledge things [1]. The notion of an item is extended here to incorporate two powerful classes of acceptability measures. The key to this formalism is the way in which the “meaning” of an item, called its “semantics”, is specified. A single rule of decomposition is specified for items [5]. Items are *either* represented informally as “i-schema” *or* formally as λ -calculus expressions. The i-schema notation is used in applications.

An M-adic item is a named triple $A[S_A, V_A, C_A]$ with *item name* A , S_A is the *item semantics* of A , V_A is the *item value acceptability measure* of A and C_A is the *item set acceptability measure* of A .

The *semantics* of an item is a function that *recognises* the members of the “value set” of that item [6]. The *value set* of an information item is the set of tuples that are associated with a relational implementation of that item. Knowledge items, including complex, recursive knowledge items, have value sets too [1]. 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 Fig. 1. Items are extended here by introducing two distinct classes of acceptability measures.

The *value acceptability* measure of an item is a fuzzy estimate of the likelihood that a given tuple is *not* in the

[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

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

item's value set. This measure is a fuzzy expression of the form:

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

where $\{A_1, \dots, A_n\}$ are the components of item A , K is a fuzzy predicate and \wedge is the fuzzy "min" conjunction.

The *set acceptability* measure of an item is a fuzzy estimate of the likelihood that the general structure of the item's value set is invalid [7]. This measure is a fuzzy expression of the form:

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

where \wedge is the fuzzy "min" conjunction and L is a fuzzy expression constructed as a logical combination of:

- $Card_A$ lies in some numerical range;
- $Uni(A_i)$ for some i , $1 \leq i \leq n$, and
- $Can(A_i, X)$ 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 ,

" $Uni(A_i)$ " is a fuzzy predicate whose truth value is "the proportion of the members of the value set of item A_i that also occur in the value set of item A ". " $Can(A_i, X)$ " is a fuzzy predicate whose truth value is "in the value set of item A , the proportion of members of the value set of the set of items X that functionally determine members of the value set of item A_i ". " $Card_A$ " means "the number of different values in the value set of item A ". The subscripts identify the item's components to which that measure applies.

For example, the semantics of the [part/sale-price, part/cost-price, mark-up] item is:

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

So an item's semantics specifies what *should be* in an implementation, and the two acceptability measures are measures of invalidity of what *is* in an implementation. If an acceptability measure does *not* detect invalidity to any level of (fuzzy) significance then this does *not* imply that an implementation is valid.

For example, an application could contain a spare-part thing that is represented by the item *part*. If spare parts are identified by their part-number then the semantics of *part* is $\lambda x \bullet [is-a[x:part-number]] \bullet$ where the function "is-a" means "x if x is in P" and "undefined otherwise". Suppose that the generally expected range for part numbers is $[0, 2000]$. Then the value acceptability measure for the item *part* could be $\lambda x \bullet [f(x)] \bullet$ where:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } 0 \leq x \leq 2000 \\ 2 - \frac{x}{2000} & \text{if } 2000 < x < 4000 \\ 0 & \text{if } x \geq 4000 \end{cases}$$

$Card_{part}$ is the number of *different* part-numbers. Suppose that the generally expected range for $Card_{part}$ is less than 100. Then the set acceptability measure, C_{part} , for the item *part* is a fuzzy predicate; it could be:

$$C_{part} = \begin{cases} 1 & \text{if } Card_{part} \leq 100 \\ 2 - \frac{Card_{part}}{100} & \text{if } 100 < Card_{part} < 200 \\ 0 & \text{if } Card_{part} \geq 200 \end{cases}$$

In this way, value and set acceptability measures are developed for data items, and similarly for information and knowledge items.

Items make it difficult to analyse the structure of the whole application because, for example, two rules that share the same basic wisdom may be expressed in terms of quite different components; this could obscure their common wisdom. To make the inherent structure of knowledge clear 'objects' are introduced as item building operators [1].

Object names are written in bold italics. Suppose that the conceptual model already contains the item "*part*" which represents spare parts, and the item "*cost-price*" which represents cost prices; then the information "spare parts have a cost price" can be represented by "*part/cost-price*" which may be built by applying the "*costs*" object to *part* and *cost-price*:

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

In [5] the decomposition of items and objects is described. Decomposition removes hidden relationships from the conceptual model. Hidden relationships can present a maintenance hazard. Here decomposition simplifies the estimation of knowledge integrity.

Degradation of Knowledge Integrity

The semantics of an item $A [S_A, V_A, C_A]$ is a function that recognises the members of its value set. The value set is a conceptual notion in the system design. So the value set of the item A —as in the definition of an M-adic item above—at time τ is:

$$\gamma^\tau(A) = \{ y_1^1 \dots y_{m_1}^1 \dots y_{m_n}^n : S_A(y_1^1, \dots, y_{m_1}^1, \dots, y_{m_n}^n) \text{ at time } \tau \}$$

A *knowledge base implementation* is a set of knowledge items and a set of stored relations and data domains

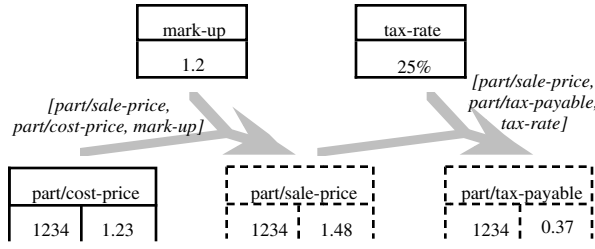


Fig. 2. Real and virtual items

representing some information and data items. Some information and data items are associated with actual stored data, and some are not. Knowledge items are not normally associated with actual stored data. If an item is associated with actual stored data then it is a *real item*; otherwise it is a *virtual item*. The set of tuples in the implementation of the *real item* A is denoted by $\lambda^\alpha(A)$ where α is the time of the most recent modification to those tuples. Knowledge items may be used to derive tuples for virtual data and information items. For example, suppose that the real data item *mark-up* has a stored data value mark-up, and that the real information item *part/cost-price* has a stored relation part/cost-price. Then the knowledge item $[part/sale-price, part/cost-price, mark-up]$ —or an “if-then” implementation of it—may be used to *derive* tuples in the relation for the virtual item *part/sale-price*. Further, the knowledge item $[part/sale-price, part/tax-payable, tax-rate]$ could then enable the tuples in the relation for the virtual item *part/tax-payable* to be derived. This is illustrated in Fig. 2. If a virtual item A_i is a component of a knowledge item A where the tuples (or data values) associated with A_i are

derived from $\{ \lambda^{\alpha_j}(A_j) : A_j \text{ is a component of } A, j \neq i \}$ using the knowledge A then A_i is *derivable* and those tuples (or data values) are called the *derived set* which is denoted by $\lambda^\beta(A_i)$ where β is the time at which the derivation is performed. This definition is recursive. In Fig. 2 only part/cost-price, mark-up and tax-rate are stored. This example shows how the validity of the calculation of the tuples in the relation for the virtual item *part/tax-payable* relies on the validity of those three real items *and* on the validity of those two knowledge items. So if the validity of any of those three real items or either of those two knowledge items has “degraded” in some way then that calculation *may* yield an incorrect result.

For a knowledge base, its implementation may not be valid because *updates* that should have been performed were not, or *modifications* that should not have been performed were. The corruption of a knowledge base by such modifications is not considered here. The failure to perform updates is considered. So *updates* are changes that should have been performed on the implementation of real items or knowledge items. In addition, incorrect values may be attributed to a knowledge base implementation because the derived tuples for some virtual items were calculated prior to required updates being performed [8]. At time α the *true set* for a—real or

$\lambda^\alpha(part)$	$\Lambda^\tau(part)$	$\lambda^\alpha(part/cost-price)$	$\Lambda^\tau(part/cost-price)$
<i>part</i>	<i>part</i>	<i>part/cost-price</i>	
part-number	part-number	part-number	dollar-amount
1234	1234	1234	1.20
2345	2345	2345	2.40
2468	2456	2468	3.60
3456	2468	3456	4.80
3579	3456	3579	5.10
4567	3579	4567	6.30
	4680		
			part-number
			dollar-amount
			1234
			1.20
			2345
			2.60
			2456
			0.60
			2468
			3.80
			3456
			4.80
			3579
			5.20
			4680
			0.80

Fig. 3. The implementation and the true set

virtual—item A is the set of tuples that should be associated with A at time α ; it is denoted by $\Lambda^\alpha(A)$. In other words, the implementation is what *is* either stored or derived in the knowledge base, the true set is what *should be* either stored or derived.

Suppose that the implementation of the real data item *part* was stored at time α . Then at a subsequent time τ the implementation and the true set may be as shown on the left side of Fig. 3. In that Figure the implementation for the data item *part* contains the part number “4567” that should *not* be there, and does *not* contain two part numbers which *should* be there. Suppose that the implementation for the real information item *part/cost-price* was stored at time α . Then at a subsequent time τ the implementation and the true set may be as shown on the right side of Fig. 3. Likewise the implementation and the true set of a knowledge item may contain tuples that should not be there, and may not contain tuples that should be there.

Fig. 2 shows a knowledge base implementation consisting of two knowledge items, three information items and two data items. If the design and implementation are correct then the tuples associated with each real or virtual item will be the same as that item’s true set.

Item and Object Validity

At time τ , the true set $\Lambda^\tau(A)$ and $\lambda^\alpha(A)$, $\alpha \leq \tau$, of item A may not be the same [9]. The implementation of a real item is “correct” as long as its tuples have been correctly stored and maintained [10]. The derived set of a virtual item is “correct” as long as the knowledge used to derive the tuples for that item has been correctly maintained *and* the stored data used by that knowledge has been correctly maintained. In reality we may hope that the implementation is correct, and expect that it is incorrect [11]. To measure the extent that the implementation or the derived set are the same as the true set, let $p_{X/Y}$ be the proportion of those elements in set X that are also in set Y . Then the *difference measure*:

$$\Delta(A, B) = \sqrt{p_{A/B} \times p_{B/A}}$$

is unity if both sets are identical and is zero if one set contains no elements from the other; the square root

ensures that this measure retains linearity with measured proportion. For example, if each set contains exactly half of the members of the other set then the value of this measure is 0.5. The value of the difference measure is *not* necessarily equal to the proportion of valid members in either set because the difference measure takes account of those elements that are not in each set but should be there. The *validity* of a real or virtual item A at time τ is:

$$\delta_A(\tau) = \Delta(\lambda^\tau(A), \Lambda^\alpha(A))$$

where $\tau \geq \alpha$ and the difference measure Δ is as defined above. $0 \leq \delta_A(\tau) \leq 1$. If $\delta_A(\tau) = 1$ then item A is *valid*. If $\delta_A(\tau) = 0$ then the set of tuples associated with item A contains no tuples that it should contain and A is *completely invalid*.

The knowledge item [*part/sale-price, part/cost-price, mark-up*] is built by applying the knowledge object **mark-up-rule** to the three items *part/sale-price, part/cost-price* and *mark-up*. That knowledge item may be used to derive tuples for the information item *part/sale-price*. The accuracy of these derived tuples relies on the implementation of the two real items *mark-up* and *part/cost-price* being accurate *and* on the knowledge object **mark-up-rule** being accurate. The integrity of these items and objects is expected to degrade as time progresses.

If we know precisely what knowledge degradation has occurred then we can usually rectify it [12]. In practice we tend to have some loose expectation ε of the validity δ . For example, we may expect that “within a year the whole *part/cost-price* relation will be out of date”. So our expectation for the validity of the *part/cost-price* item may be represented by a function with a linear decay of one year’s extent. Also, we may expect that “as the ‘types’ of parts are redesignated, the integrity of the *part/type* relation will degrade decreasingly over time so that in a year roughly half of the relation will be out of date and in a ‘very long time’ the whole relation will be out of date”. So our expectation for the degradation of the *part/type* item may be represented by a function with an exponential decay with a half-life of one year. The validity estimates for these two examples are:

$$\varepsilon_{part/cost-price}(\tau) = \begin{cases} 1 - \tau & \text{if } 0 \leq \tau \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\varepsilon_{part/type}(\tau) = 2^{-\tau} \text{ for } \tau \geq 0$$

The validity of an object will contribute to the validity of any item generated by that object. But objects do not have a value set or an implementation. The validity of the item A , generated using object B , $A = B(C, D, E, F)$ may be analysed completely in terms of the validity of items C, D, E and F and the validity of object B . In other words, if items C, D, E and F are valid then the validity of item A will be attributable entirely to the validity of object B . So the *validity* of an object is the validity of any item

generated by that object when applied to a set of valid items.

Propagating Validity Estimates

It would be convenient if:

$$\varepsilon_{ob(cc, dd)}(\tau) = \varepsilon_{ob}(\tau) \times \varepsilon_{cc}(\tau) \times \varepsilon_{dd}(\tau)$$

but this product rule is *not* valid in general because the validity of the components—in the above example the components are *cc* and *dd*—may be logically dependent. This product rule is valid if the validity of the object **ob** and the validity of the two component items are all independent. If the validity of X is independent of the validity of Y then $(\varepsilon_X(\tau) | \varepsilon_Y(\tau)) = \varepsilon_X(\tau)$. If the validity of X is determined by the validity of Y then $(\varepsilon_X(\tau) | \varepsilon_Y(\tau)) = \varepsilon_Y(\tau)$.

Suppose that object **ob** is applied to a set of n component items $C = \{cc, D\}$ where *cc* is a component item and D is a set of $n-1$ component items. The general rule for propagating validity estimates through an object operator is:

$$\begin{aligned} \models \varepsilon_{ob(C)}(\tau) &= \varepsilon_{ob}(\tau) \times (\varepsilon_C(\tau) | \varepsilon_{ob}(\tau)) \\ &= \varepsilon_C(\tau) \times (\varepsilon_{ob}(\tau) | \varepsilon_C(\tau)) \end{aligned}$$

where $(\varepsilon_C(\tau) | \varepsilon_{ob}(\tau))$ is “an estimate of the validity of the set of n component items C at time τ given that the estimate of the validity of object **ob** at time τ is $\varepsilon_{ob}(\tau)$.” If the conceptual model has been decomposed then $(\varepsilon_C(\tau) | \varepsilon_{ob}(\tau)) = \varepsilon_C(\tau)$. To propagate validity estimates across a set of component items, suppose that the set C is a set of n component items as above. The general recursive rule for propagating validity estimates over such a set is:

$$\models \varepsilon_{\{cc, D\}}(\tau) = \varepsilon_{cc}(\tau) \times (\varepsilon_D(\tau) | \varepsilon_{cc}(\tau))$$

where $(\varepsilon_D(\tau) | \varepsilon_{cc}(\tau))$ is “an estimate of the validity of the set of $n-1$ component items D at time τ given that the estimate of the validity of the component item *cc* at time τ is $\varepsilon_{cc}(\tau)$.” In general $(\varepsilon_D(\tau) | \varepsilon_{cc}(\tau)) \neq \varepsilon_D(\tau)$ even if the conceptual model has been decomposed. Now suppose that item *cc* is virtual and that it is derivable from the set of items D using knowledge item **ob**(*cc, D*), ie that $\text{Can}(cc, D) = 1$ in **ob**(*cc, D*). Then the validity of item *cc* depends on the validity of object **ob** and on the validity of the items in the set D . Estimates of these validities are propagated by:

$$\models \varepsilon_{cc}(\tau) = \varepsilon_{ob}(\tau) \times (\varepsilon_D(\tau) | \varepsilon_{ob}(\tau))$$

If the conceptual model has been decomposed then $(\varepsilon_D(\tau) | \varepsilon_{ob}(\tau)) = \varepsilon_D(\tau)$.

The three rules above for propagating validity estimates follow from the definition of item validity. These three

rules lead to complex expressions for validity estimates due to the quantity of conditional expressions—ie expressions involving “|”. But the quantity of these conditional expressions may be reduced.

If the knowledge base has been decomposed [1] and if the sub-item relationships have been reduced to sub-type relationships [4] between data items then the object operators and the basis items in the conceptual model are independent with the possible exception of sub-item relationships between data items [1]. So if the knowledge base has been decomposed and there are no sub-item relationships between data items then all the conditional expressions may be removed. For example, if the knowledge base has been decomposed and there are no sub-item relationships between data items then the validity estimate for the virtual information item *part/tax-payable* is:

$$\begin{aligned}
\varepsilon_{part/tax-payable}(\tau) &= \varepsilon_{tax-rule}(\tau) \\
&\quad \times (\varepsilon_{\{part/sale-price, tax-rate\}}(\tau) | \varepsilon_{tax-rule}(\tau)) \\
&= \varepsilon_{tax-rule}(\tau) \times \varepsilon_{\{part/sale-price, tax-rate\}}(\tau) \\
&= \varepsilon_{tax-rule}(\tau) \times \varepsilon_{part/sale-price}(\tau) \\
&\quad \times (\varepsilon_{tax-rate}(\tau) | \varepsilon_{part/sale-price}(\tau)) \\
&= \varepsilon_{tax-rule}(\tau) \times \varepsilon_{part/sale-price}(\tau) \times \varepsilon_{tax-rate}(\tau) \\
&= \varepsilon_{tax-rule}(\tau) \times \varepsilon_{mark-up-rule}(\tau) \\
&\quad \times (\varepsilon_{\{part/cost-price, mark-up\}}(\tau) | \varepsilon_{mark-up-rule}(\tau)) \\
&\quad \times \varepsilon_{tax-rate}(\tau) \\
&= \varepsilon_{tax-rule}(\tau) \times \varepsilon_{mark-up-rule}(\tau) \\
&\quad \times \varepsilon_{part/cost-price}(\tau) \times \varepsilon_{mark-up}(\tau) \times \varepsilon_{tax-rate}(\tau)
\end{aligned}$$

If sub-item relationships are present or if the conceptual model has not been decomposed then the calculations become more involved. For example, suppose the *supervisor* data item is a sub-item of the *person* data item. Consider the real *person/supervisor* item; the implementation of which is populated with 2-tuples (person-id, person-id) where the second person is the “supervisor” of the first. Suppose that this item is built by applying the **super** information object to the data items *person* and *supervisor*:

$$person/supervisor = \mathbf{super}(person, supervisor)$$

then the validity estimate of the *person/supervisor* information item will be:

$$\begin{aligned}
\varepsilon_{\mathbf{super}(person, supervisor)}(\tau) \\
&= \varepsilon_{\{person, supervisor\}}(\tau) \\
&\quad \times (\varepsilon_{\mathbf{super}}(\tau) | \varepsilon_{\{
\end{aligned}$$