Knowledge Object Decomposition

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 conceptual model for knowledge bases the data and knowledge are represented formally as "items". The foundation of this model is a "basis" consisting of data items. "Objects" are "item building" operators that are applied to the basis to build the information items and knowledge items in the conceptual model. Objects may be decomposed. In this way the knowledge in the resulting conceptual model is simplified. Knowledge object decomposition removes hidden and unwanted relationships from the knowledge base.

Introduction

The majority of conceptual models treat the "rule base" component [1] separately from the "database" component, [2] and [3]. This enables well established design methodologies to be employed, but the use of two separate models means that the interrelationship between the things in these two models cannot be represented, integrated and manipulated naturally [4]. Further, neither of these separate models is therefore able to address completely the maintenance [5] of the whole knowledge base [6].

The first step in a design methodology [7] is the construction of a *requirements model*, and the second step is the construction of a *conceptual model* that specifies *how* the system will do what it is required to do. The conceptual model is expressed in terms of items and objects. The third step in that design methodology is the construction of an *external model* [8] that shows how the knowledge in the conceptual model may deliver the functionality required. The external model is also expressed in terms of items and objects.

The conceptual model described in [9] is built by applying "objects" as operators to data "items" in the model "basis". Items and objects enable a uniform approach to be taken to knowledge representation. Items and objects are viewed either formally as λ-calculus expressions or informally as schema. The λ-calculus view provides a sound theoretical basis for the work; it is *not* intended for practical use. So the approach has both theoretical and practical significance. If the object operators are decomposed then hidden and unwanted relationships in the knowledge are removed and so the resulting conceptual model is simplified.

Objects

In [9] items are defined formally, and objects are introduced informally to build the conceptual model. Here objects and their decomposition are defined.

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 $\mathbf{X}^m$. Formally, given a unique name $\boldsymbol{A}$, a j-tuple $(i_1, i_2,..., i_j)$, j variables $(Q_1, Q_2,..., Q_j)$ of type $(X_1^{i_1}, X_2^{i_2},..., X_j^{i_j})$ respectively where each X_k represents a type such as $\mathbf{X}$, $\mathbf{D}$, $\mathbf{I}$ or $\mathbf{K}$, an n-tuple $(m_1, m_2,..., m_n)$, $n \geq j$, $M = \Sigma_i m_i$, n not necessarily distinct variables $(P_1, P_2,..., P_n)$ such that $(\exists x: 1 \leq x \leq n)(\forall y)(P_x = Q_y \text{ and } m_x = i_y)$:

- E is a j-argument typed λ-calculus expression of type $(X_1^{i_1}, X_2^{i_2},..., X_j^{i_j})$:

  $$\lambda Q_1{:}X_1^{i_1} Q_2{:}X_2^{i_2} ... Q_j{:}X_j^{i_j} \bullet \lambda y_1^1 ... y_{m_1}^1 ... y_{m_n}^n \bullet [S_{P_1}(y_1^1,...,y_{m_1}^1) \circ S_{P_2}(y_1^2,...,y_{m_2}^2) \circ ..... \circ S_{P_n}(y_1^n,...,y_{m_n}^n) \circ J(y_1^1 ... y_{m_1}^1 ... y_{m_n}^n)] \bullet\bullet$$

  where S_B is the semantics of item *B*, and if J is a "separable" predicate as defined on the following page then it should be shown in its separated form. S_B is a λ-calculus expression that recognises the members of the value set of item *B* as defined in [9].

- F is a j-argument typed λ-calculus expression of type $(X_1^{i_1}, X_2^{i_2},..., X_j^{i_j})$:

  $$\lambda Q_1{:}X_1^{i_1} Q_2{:}X_2^{i_2} ... Q_j{:}X_j^{i_j} \bullet \lambda y_1^1 ... y_{m_1}^1 ... y_{m_n}^n \bullet [V_{P_1}(y_1^1,...,y_{m_1}^1) \circ V_{P_2}(y_1^2,...,y_{m_2}^2) \circ ..... \circ V_{P_n}(y_1^n,...,y_{m_n}^n) \circ K(y_1^1,...y_{m_1}^1 ... y_{m_n}^n)] \bullet\bullet$$

  where V_B denotes the value constraints of item *B*, and K is essentially an m-argument predicate where $\min(M,2) \leq m \leq M$. V_B is a λ-calculus expression that is satisfied by the members of the value set of item *B* as defined in [9].

- G is a j-argument typed λ-calculus expression of type $(X_1^{i_1}, X_2^{i_2},..., X_j^{i_j})$:

  $\lambda Q_1{:}X_1^{i_1} Q_2{:}X_2^{i_2}...Q_j{:}X_j^{i_j}\bullet[C_{Q_1} \circ ..... \circ C_{Q_j} \circ (L(Q_1,..,Q_j))_{\Psi(\boldsymbol{A},Q_1,..,Q_j)}]\bullet$

  where C_B denotes the item set constraints of item *B* as defined in [9]. $\Psi(\boldsymbol{A},Q_1,..,Q_j)$ is the name of the item that results from the application of object ***A*** to items $\{Q_1,..,Q_j\}$. L consists of a logical combination of:

  - Card lies in some numerical range;
  - $Uni(Q_k)$ for some k, $1 \le k \le n$, and
  - $Can(Q_k, X)$ for some k, $1 \le k \le j$, where X is a non-empty subset of $\{Q_1,Q_2,...,Q_j\} - \{Q_k\}$

which are as defined in [9] then the named triple:

A[E,F,G]

is an M-adic *object* with *object name* ***A***, of argument type $(X_1^{i_1}, X_2^{i_2},..., X_j^{i_j})$. E is the *object semantics* of ***A***. F is the *object value constraints* of ***A***. G is the *object set constraints* of ***A***. "Uni(*a*)" means that "all members of the value set of item *a* must be in this association". "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.

The definition of an object refers to "separable" predicates [10]. Given a predicate J of the form:

$J(y_1^1,...,y_{m_1}^1,y_1^2,...,y_{m_2}^2,.....,y_1^n,...,y_{m_n}^n)$

Define the set $\{Y_1, Y_2,..., Y_n\}$ by $Y_i = \{y_1^i,...,y_{m_i}^i\}$. If J can be written in the form:

$J_1 \circ J_2 \circ ... \circ J_m$

where each J_i is a predicate in terms of the set of variables X_i with:

$X_i \subset Y_1 \cup Y_2 \cup ... \cup Y_n$, and

for each X_i (ßj) such that X_i does *not* contain any of the variables in Y_j then predicate J is *separable* into the partition $\{X_1, X_2,..., X_m\}$.

There are a number of different senses in which two objects can be considered to be "equal". Three of these senses of "object equality" are described now. Given two n-adic objects of identical argument type $(X^{m_1}, X^{m_2},..., X^{m_n})$ where each X is a type such as **X**, **K**, **I** or **D** (standing respectively for "free", "knowledge", "information" or "data"):

$\boldsymbol{A}[E_{\boldsymbol{A}}, F_{\boldsymbol{A}}, G_{\boldsymbol{A}}]$ and
$\boldsymbol{B}[E_{\boldsymbol{B}}, F_{\boldsymbol{B}}, G_{\boldsymbol{B}}]$

A and ***B*** are *identical*, written $\boldsymbol{A} \equiv \boldsymbol{B}$, if:

$(\forall Q_1,Q_2,...,Q_j)[E_{\boldsymbol{A}}(Q_1,Q_2,...,Q_j) \leftrightarrow E_{\boldsymbol{B}}(Q_1,Q_2,...,Q_j)]$
$(\forall Q_1,Q_2,...,Q_j)[F_{\boldsymbol{A}}(Q_1,Q_2,...,Q_j) \leftrightarrow F_{\boldsymbol{B}}(Q_1,Q_2,...,Q_j)]$
$(\forall Q_1,Q_2,...,Q_j)[G_{\boldsymbol{A}}(Q_1,Q_2,...,Q_j) \leftrightarrow G_{\boldsymbol{B}}(Q_1,Q_2,...,Q_j)]$

If two objects are identical then they will not necessarily have the same name.

A and ***B*** are *equivalent*, written $\boldsymbol{A} \simeq \boldsymbol{B}$, if they are both of the same argument type and there exists a permutation π such that:

$(\forall Q_1,Q_2,...,Q_j)[E_{\boldsymbol{A}}(Q_1,Q_2,...,Q_j) \leftrightarrow E_{\boldsymbol{B}}(\pi(Q_1,Q_2,...,Q_j))]$
$(\forall Q_1,Q_2,...,Q_j)[F_{\boldsymbol{A}}(Q_1,Q_2,...,Q_j) \leftrightarrow F_{\boldsymbol{B}}(\pi(Q_1,Q_2,...,Q_j))]$
$(\forall Q_1,Q_2,...,Q_j)[G_{\boldsymbol{A}}(Q_1,Q_2,...,Q_j) \leftrightarrow G_{\boldsymbol{B}}(\pi(Q_1,Q_2,...,Q_j))]$

A and ***B*** are *weakly equivalent*, written $\boldsymbol{A} \simeq_w \boldsymbol{B}$, if they are both of the same argument type and there exists a permutation π such that:

$(\forall Q_1,Q_2,...,Q_j)[E_{\boldsymbol{A}}(Q_1,Q_2,...,Q_j) \leftrightarrow E_{\boldsymbol{B}}(\pi(Q_1,Q_2,...,Q_j))]$

As an example of the application of an object operator, the *part/cost-price* item can be built from the items *part* and *cost-price* using the ***costs*** operator:

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

$\boldsymbol{costs}[\lambda P{:}\mathbf{X}^1 Q{:}\mathbf{X}^1\bullet\lambda xy\bullet[Sp(x) \circ S_Q(y) \circ costs(x,y)]\bullet\bullet,$
$\lambda P{:}\mathbf{X}^1 Q{:}\mathbf{X}^1\bullet\lambda xy\bullet[V_P(x) \circ V_Q(y) \circ$
$((1000 < x < 1999) \rightarrow (y \le 300))]\bullet\bullet,$
$\lambda P{:}\mathbf{X}^1 Q{:}\mathbf{X}^1\bullet[C_P \circ C_Q$
$\circ (Uni(P) \circ Can(Q, \{P\}))_{\Psi(\boldsymbol{costs},P,Q)}]\bullet]$

where $\Psi(\boldsymbol{costs}, P, Q)$ is the name of the item ***costs***(P, Q).

Data objects provide a representation of sub-typing. Items are a universal formalism for representation but make it difficult to analyse the structure of the whole application. 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 items clear 'objects' are introduced as item building operators. For example, consider the *[part/sale-price, part/cost-price, 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 $(\mathbf{I}^2, \mathbf{I}^2, \mathbf{D}^1)$ to the items *part/sale-price*, *part/cost-price* and *mark-up*. That is:

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

Objects also represent value constraints and set constraints in a uniform way. A "join" operation for items is defined in [4].

A *conceptual model* consists of:

- a *basis*, which is a fundamental set of data items on which the conceptual model is founded;
- an *object library*, that is a set of object operators which are used to construct the items in the conceptual model with the exception of the items in the basis;

- a *conceptual diagram*, that shows how the objects in the object library are used to construct the items in the conceptual model, and
- *maintenance links* that join two items in the conceptual diagram if modification to one of them necessarily means that the other has at least to be checked for correctness if validity is to be preserved.

The conceptual diagram provides a convenient, high-level view of the conceptual model.

Object Decomposition

Object decomposition removes hidden and unwanted relationships from knowledge [11]. Given two objects:

$\boldsymbol{A}[\ E_{\boldsymbol{A}}, F_{\boldsymbol{A}}, G_{\boldsymbol{A}}\ ]$ and

$\boldsymbol{B}[\ E_{\boldsymbol{B}}, F_{\boldsymbol{B}}, G_{\boldsymbol{B}}\ ]$

then $E_{\boldsymbol{A}}$, $F_{\boldsymbol{A}}$, $E_{\boldsymbol{B}}$ and $F_{\boldsymbol{B}}$ all have the form:

λ<string of typed outer variables>•λ<string of inner variables>•[...]••

Both $G_{\boldsymbol{A}}$ and $G_{\boldsymbol{B}}$ have the form:

λ<string of typed outer variables>•[...]•

Suppose that $E_{\boldsymbol{A}}$ has n typed outer variables, and that $E_{\boldsymbol{B}}$ has m typed outer variables. Some of the outer arguments of $\boldsymbol{A}$ and $\boldsymbol{B}$ may have identical argument types. Suppose that k pairs of outer arguments of $\boldsymbol{A}$ and $\boldsymbol{B}$ that have identical argument types are identified, where $k \geq 0$. Let C be a set of k pairs of indices where the first index in each pair identifies an outer variable of $\boldsymbol{A}$ and the second identifies an outer variable of $\boldsymbol{B}$ of identical type. C may be empty. To ensure that C is well defined the pairs of indices in C occur in ascending order of the first index of each pair. Let $\boldsymbol{A}^*$ be an object that is identical to object $\boldsymbol{A}$ except for the order of its outer and inner variables. The last k outer variables in $\boldsymbol{A}^*$ are those outer variables in $\boldsymbol{A}$ that are referred to by the indices of $\boldsymbol{A}$ that are in the set C. The inner variables of $\boldsymbol{A}^*$ are a permutation of the inner variables of $\boldsymbol{A}$ to ensure that $\boldsymbol{A}^*$ is identical to $\boldsymbol{A}$, except for the order of its variables. Let $\boldsymbol{B}^*$ be an object that is identical to object $\boldsymbol{B}$ except for the order of its outer and inner variables. The first k outer variables in $\boldsymbol{B}^*$ are those outer variables in $\boldsymbol{B}$ that are referred to by the indices of $\boldsymbol{B}$ that are in the set C. The inner variables of $\boldsymbol{B}^*$ are a permutation of the inner variables of $\boldsymbol{B}$ to ensure that $\boldsymbol{B}^*$ is identical to $\boldsymbol{B}$, except for the order of its variables. Let π be a permutation that turns the ordered set of outer variables of $\boldsymbol{A}^*$ into the ordered set of outer variables of $\boldsymbol{A}$. Let π′ be a permutation that turns the ordered set of outer variables of $\boldsymbol{B}^*$ into the ordered set of outer variables of $\boldsymbol{B}$. Let ρ be a permutation that turns the ordered set of inner variables of $\boldsymbol{A}^*$ into the ordered set of inner variables of $\boldsymbol{A}$. Let ρ' be a permutation that turns the ordered set of inner variables of $\boldsymbol{B}^*$ into the ordered set of inner variables of $\boldsymbol{B}$. Suppose that 'P' is an (n - k)-tuple of typed variables, 'Q' is a k-tuple of typed variables and 'R' is an (m - k)-tuple of typed variables. Suppose that 'x' is a string of inner variables that correspond to 'P', 'y' is a string of inner variables that correspond to 'Q' and 'z' is a string of inner variables that correspond to 'R'. Then the object with name $\boldsymbol{A}\ {}_C\ \boldsymbol{B}$ is the *join* of $\boldsymbol{A}$ and $\boldsymbol{B}$ on C and is defined to be:

$(\boldsymbol{A}\ {}_C\ \boldsymbol{B})[$

$\lambda P{:}\mathbf{T}_P Q{:}\mathbf{T}_Q R{:}\mathbf{T}_R \bullet \lambda xyz \bullet [E_{\boldsymbol{A}}(\pi(P,Q))(\rho(x,y)) \circ E_{\boldsymbol{B}}(\pi'(Q,R))(\rho'(x,y))]\bullet,$

$\lambda P{:}\mathbf{T}_P Q{:}\mathbf{T}_Q R{:}\mathbf{T}_R \bullet \lambda xyz \bullet [F_{\boldsymbol{A}}(\pi(P,Q))(\rho(x,y)) \circ F_{\boldsymbol{B}}(\pi'(Q,R))(\rho'(x,y))]\bullet,$

$[G_{\boldsymbol{A}}(\rho(P,Q)) \circ G_{\boldsymbol{B}}(\rho'(Q,R))]\bullet\ ]$

If $\boldsymbol{A}$ and $\boldsymbol{B}$ are two information objects with a single shared argument in the set C then $\boldsymbol{A}\ {}_C\ \boldsymbol{B}$ is their "join", in the conventional sense, on the domain represented by the single shared argument. If $\boldsymbol{A}$ and $\boldsymbol{B}$ are two functional associations and if C represents both the argument of one of these functions and the range of the other then $\boldsymbol{A}\ {}_C\ \boldsymbol{B}$ is the functional composition of these two functions. C may be empty. If C = Á then $\boldsymbol{A}\ {}_C\ \boldsymbol{B}$ is the Cartesian product of $\boldsymbol{A}$ and $\boldsymbol{B}$.

If two objects $\boldsymbol{A}$ and $\boldsymbol{B}$ are such that for each argument of $\boldsymbol{A}$ there exists an argument of $\boldsymbol{B}$ with identical type and object $\boldsymbol{B}$ has at least one argument that is not an argument of object $\boldsymbol{A}$, and $\boldsymbol{A}\ {}_C\ \boldsymbol{B} = \boldsymbol{B}$, then object $\boldsymbol{A}$ is a *sub-object* of object $\boldsymbol{B}$, written $\boldsymbol{A} \subset \boldsymbol{B}$. The composition $\boldsymbol{A}\ {}_C\ \boldsymbol{B}$ is a *monotonic composition* if $\boldsymbol{A}\ {}_C\ \boldsymbol{B}$ is not weakly equivalent with either $\boldsymbol{A}$ or $\boldsymbol{B}$. If $\boldsymbol{A}\ {}_C\ \boldsymbol{B}$ is a monotonic composition and the set C identifies one argument only of $\boldsymbol{A}$ and $\boldsymbol{B}$ then $\boldsymbol{A}\ {}_C\ \boldsymbol{B}$ is a *unit composition*.

The following are properties of :

- $\boldsymbol{A}\ {}_K\ \boldsymbol{A} = \boldsymbol{A}$ where $K \subseteq K_{\boldsymbol{A}}$ and $K_{\boldsymbol{A}}$ is the set of arguments of $\boldsymbol{A}$
- $\boldsymbol{A}\ {}_K\ \boldsymbol{B} \simeq \boldsymbol{B}\ {}_K\ \boldsymbol{A}$ where $K \subseteq (K_{\boldsymbol{A}} \cap K_{\boldsymbol{B}})$
- $\boldsymbol{A}\ {}_K(\boldsymbol{B}\ {}_L\ \boldsymbol{C}) \simeq (\boldsymbol{A}\ {}_K\ \boldsymbol{B})\ {}_L\ \boldsymbol{C}$ where $K \subseteq (K_{\boldsymbol{A}} \cap K_{\boldsymbol{B}})$ and $L \subseteq (K_{\boldsymbol{B}} \cap K_{\boldsymbol{C}})$

If $\boldsymbol{A}\ {}_K\ \boldsymbol{B} = \boldsymbol{A}$ then $K = K_{\boldsymbol{B}} \subseteq K_{\boldsymbol{A}}$.

Using the rule of composition , knowledge objects, information objects and data objects may be combined with one another. Object $\boldsymbol{O}$ is *decomposable* into $\boldsymbol{D} = \{\boldsymbol{O}_1, \boldsymbol{O}_2, .., \boldsymbol{O}_n\}$ if

- $\boldsymbol{O}_i$ is not tautological for all i,
- $\boldsymbol{O} = \boldsymbol{O}_1\ \ \boldsymbol{O}_2\ \ ...\ \ \boldsymbol{O}_n$, where
- each composition is monotonic

If object $\boldsymbol{O}$ is decomposable then it does not necessarily have a unique decomposition.

Principles for Decomposition

Principles for decomposition assist with the recognition of decomposable objects. For example, object decomposition enables:

$$\textbf{\textit{two-type}}[\ \lambda P{:}\mathbf{D}^1 Q{:}\mathbf{D}^1 R{:}\mathbf{X}^1 \bullet \lambda xyz \bullet [\ S_P(x) \circ S_Q(y) \circ S_R(z) \circ \text{has-type}(y, z)\ ]\bullet\bullet,\ \lambda P{:}\mathbf{D}^1 Q{:}\mathbf{D}^1 R{:}\mathbf{X}^1 \bullet \lambda xyz \bullet [\ V_P(x) \circ V_Q(y) \circ V_R(z)\ ]\bullet\bullet,\ \lambda P{:}\mathbf{D}^1 Q{:}\mathbf{D}^1 R{:}\mathbf{X}^1 \bullet [C_P \circ C_Q \circ C_R\ (\text{Uni}(P) \circ \text{Uni}(Q) \circ \text{Can}(R, \{Q\}))\Downarrow_{(\textbf{\textit{two-type}},P,Q)}]\bullet]$$

to be decomposed into the data object:

$$\textbf{\textit{comp}}[\ \lambda P{:}\mathbf{D}^1 Q{:}\mathbf{D}^1 \bullet \lambda xy \bullet [\ S_P(x) \circ S_Q(y)\ ]\bullet\bullet,\ \lambda P{:}\mathbf{D}^1 Q{:}\mathbf{D}^1 \bullet \lambda xy \bullet [\ V_P(x) \circ V_Q(y)\ ]\bullet\bullet,\ \lambda P{:}\mathbf{D}^1 Q{:}\mathbf{D}^1 \bullet [C_P \circ C_Q \circ (\text{Uni}(P) \circ \text{Uni}(Q))\Downarrow_{(\textbf{\textit{comp}},P,Q)}]\bullet\ ]$$

and the information object:

$$\textbf{\textit{has-type}}[\lambda P{:}\mathbf{X}^1 Q{:}\mathbf{X}^1 \bullet \lambda xy \bullet [\ S_P(x) \circ S_Q(y) \circ \text{has-type}(x, y)\ ]\bullet\bullet,\ \lambda P{:}\mathbf{X}^1 Q{:}\mathbf{X}^1 \bullet \lambda xy \bullet [\ V_P(x) \circ V_Q(y)\ ]\bullet\bullet,\ \lambda P{:}\mathbf{X}^1 Q{:}\mathbf{X}^1 \bullet [C_P \circ C_Q \circ (\text{Uni}(P) \circ \text{Can}(Q, \{P\}))\Downarrow_{(\textbf{\textit{has-type}},P,Q)}\ ]\bullet]$$

The recognition of decomposable objects may not be difficult. If a given object is decomposable then its semantics permits that object to be represented as the join of two other objects. A decomposable object's semantics may be phrased so as to hide the fact that the object is decomposable. A principle for identifying decomposable objects is to examine an object's semantics expression and to note the extent to which the predicate in that expression is separable. An object's semantics is an expression of the form:

$$\lambda Q_1{:}X_1^{i_1} Q_2{:}X_2^{i_2} \ldots Q_j{:}X_j^{i_j} \bullet \lambda y_1^1 \ldots y_{m_1}^1 \ldots y_{m_n}^n \bullet [\ S_{P_1}(y_1^1,\ldots,y_{m_1}^1) \circ S_{P_2}(y_1^2,\ldots,y_{m_2}^2) \circ \ldots\ldots\ldots \circ S_{P_n}(y_1^n,\ldots,y_{m_n}^n) \circ J(y_1^1 \ldots y_{m_1}^1 \ldots y_{m_n}^n)\ ]\bullet\bullet$$

where the X_k are one of **X**, **D**, **I** or **K** for $1 \le k \le j$.

Principle 1. If the predicate in an object's semantics is separable in the sense defined above then investigate whether that object is decomposable into objects containing the argument sets identified by the separability of that predicate.

The semantics of the ***two-type*** object is:

$$\lambda P{:}\mathbf{D}^1 Q{:}\mathbf{D}^1 R{:}\mathbf{X}^1 \bullet \lambda xyz \bullet [\ S_P(x) \circ S_Q(y) \circ S_R(z) \circ \text{has-type}(y, z)\ ]\bullet\bullet$$

The predicate in this expression is separable into the partition $\{(x, y), (y, z)\}$ because:

$J_1(x) = \mathbf{T}(x, y)$

$J_2(y, z) = \text{has-type}(y, z)$

where **T** is the constant true predicate. If the predicate in an object's semantics is separable into a partition then the next step is determine whether the entire object is decomposable using the argument set indices identified by that partition. As is shown above:

$$\textbf{\textit{two-type}} = \textbf{\textit{comp}}_{\{(2,1)\}}\ \textbf{\textit{has-type}}$$

Another decomposition may be derived from the partition $\{(y, z)\}$ because $J_1(y, z) = \text{has-type}(y, z)$. This decomposition is simpler than the decomposition above.

If the predicate in an object's semantics is separable then this does not necessarily mean that that object is decomposable. So when looking for decomposable objects it is useful to have decomposition principles that are based on other parts of an object's specification besides its semantics. The remainder of this section discusses other principles that are based on the object's set constraints.

There are two different types of object. This distinction is based on whether or not an object represents an "association". If an object does *not* represent an association then that object is a data object. If an object *does* represent an association then that object is either an information object or a knowledge object [12]. If an object represents an association and that association is functional then that object can be classified by the structure of the candidate constraints that occur within its set constraints. The set constraint Can(Q, {P}) means that the value set of {P} functionally determines the value set of Q.

Object decomposition applies to all objects. If an object is an information object or a knowledge object then it represents an association. Associations are sometimes functional. Associations in knowledge can be represented as "if-then rules". Associations in information can be represented as relations with a key. No distinction is drawn here between information and knowledge functional associations. So the following principles can be employed, for example, to decompose an information object using a knowledge object or vice versa.

Objects are classified below according to the structure of the candidate constraints within their set constraints. X, Y and Z are the names of argument sets. Suppose that:

$X = \{X_1, X_2,.., X_p\}$

$Y = \{Y_1, Y_2,.., Y_q\}$

$Z = \{Z_1, Z_2,.., Z_r\}$

Suppose that object ***A*** has the two argument sets X and Y. Object ***A*** is then denoted by ***A***(X, Y). The notation $X \Rightarrow Y$ denotes that:

$\{\ \text{Can}(X_i, \{Y_1, Y_2,.., Y_q\})$: for all $i = 1,..,p\ \}$

The notation $C_{\boldsymbol{A}}[G_1, G_2,.., G_k]$ where each G_i is a term of the form $Y \Rightarrow X$ denotes that the set $\{G_1, G_2,.., G_k\}$ consists of *all* of the valid candidate constraints on object ***A***. Suppose object ***C*** is decomposable by:

$$\boldsymbol{C}(X, Y, Z) = \boldsymbol{A}(X, Y)\ _{\{(2,1)\}}\ \boldsymbol{B}(Y, Z)$$

where $C_{\boldsymbol{A}}[X \Rightarrow Y]$ and $C_{\boldsymbol{B}}[Y \Rightarrow Z]$. Principle 2 is derived by applying object decomposition to this single join decomposition.

Principle 2. Given object ***C***, if the objects ***A*** and ***B*** are not tautological, and the argument sets X, Y and Z all non-empty with:

$C_{\boldsymbol{C}}[X \Rightarrow Y, Y \Rightarrow Z]$

$C_{\boldsymbol{A}}[X \rightarrow Y]$
$C_{\boldsymbol{B}}[Y \rightarrow Z]$

then check whether:

$\boldsymbol{C}(X, Y, Z) = \boldsymbol{A}(X, Y) \underset{\{(2,1)\}}{} \boldsymbol{B}(Y, Z)$

If it does then discard object ***C*** in favour of ***A*** and ***B***.

The wisdom behind this principle is that object ***C*** contains objects ***A*** and ***B*** implicitly embedded within it. A special case of Principle 2 is derived by letting Z be Y ∪ W and object ***C*** becomes $C_{\boldsymbol{C}}[X \rightarrow (Y, W)]$ and object ***B*** becomes $C_{\boldsymbol{B}}[Á]$. The functional structure of this special case generalises the structure of the second classical normal form [13].

Principle 3. Given object ***C***, if the objects ***A*** and ***B*** are not tautological, and the argument sets X, Y and W all non-empty and:

$C_{\boldsymbol{C}}[X \rightarrow (Y, W)]$
$C_{\boldsymbol{A}}[X \rightarrow Y]$
$C_{\boldsymbol{B}}[Á]$

then check whether:

$\boldsymbol{C}(X, Z) = \boldsymbol{A}(X, Y) \underset{\{(2,1)\}}{} \boldsymbol{B}(Y, W)$

If it does then discard object ***C*** in favour of ***A*** and ***B***.

When applied to information, Principle 3 is the second classical normal form. Consider the special case when the semantics of object ***A*** is equivalent to the semantics of object ***C***. In this special case Principle 3 reduces to "...then discard object ***C*** in favour of object ***A***.". The application of this special case is illustrated below.

Principle 3 applies equally to knowledge, to a mixture of knowledge and information, and to information alone. An example of the application of this principle to knowledge follows.

Principle 4. Given object ***C***, if the objects ***A*** and ***B*** are not tautological, and the argument sets V, W, X, Y and Z all non-empty with W ⊇ V ∪ X and:

$C_{\boldsymbol{C}}[Z \rightarrow W]$
$C_{\boldsymbol{A}}[Z \rightarrow V, Y]$
$C_{\boldsymbol{B}}[Y \rightarrow X]$

then check whether:

$\boldsymbol{C}(Z, W) = \boldsymbol{A}(Y, Z, V) \underset{\{(1,1)\}}{} \boldsymbol{B}(Y, X)$

If it does then discard object ***C*** in favour of ***A*** and ***B***.

Principle 4 can be applied equally to information or to knowledge. The wisdom behind this principle for decomposition is that object ***C*** contains objects ***A*** and ***B*** implicitly embedded within it. If either objects ***A*** or ***B*** are modified then, to preserve consistency, object ***C*** would have to be modified as well. Objects ***A*** and ***B*** alone may not suffer from this difficulty.

The functional structure of Principle 4 generalises the structure of the classical third normal form. If V = Á and W = X in Principle 4 then the structure of that principle reduces to:

$C_{\boldsymbol{C}}[Z \rightarrow X]$
$C_{\boldsymbol{A}}[Z \rightarrow Y]$
$C_{\boldsymbol{B}}[Y \rightarrow X]$

Normalisation can be applied to data, information or knowledge objects, or to any combination of these. If the object operators are decomposed then hidden and unwanted relationships in the knowledge are removed and so the resulting conceptual model is simplified.

References

1. Kowalski, R.A., "Logic Programming in Artificial Intelligence", in *proceedings International Joint Conference on Artificial Intelligence*, Sydney, August 1991.
2. Lehner, F., Hofman, H.F., Setzer, R. and Maier, R., "Maintenance of Knowledge Bases", in *proceedings Fourth International Conference DEXA93*, Prague, September 1993, pp436-447.
3. J.K. Debenham, "Representing Knowledge Normalisation", *in proceedings Tenth International Conference on Software Engineering and Knowledge Engineering SEKE'98*, San Francisco, US, June 1998.
4. J.K. Debenham, "From Conceptual Model to Internal Model", in *proceedings Tenth International Symposium on Methodologies for Intelligent Systems ISMIS'97*, Charlotte, October 1997, pp227-236.
5. Compton, P., Srinivasan, A., Edwards, G., Malor, R. & Lazarus, L., "Knowledge Base Maintenance without a Knowledge Engineer", in *proceedings Expert Systems World Congress*, J. Liebowitz (Ed), Pergamon Press 1991.
6. 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.
7. Debenham, J.K., "Knowledge Simplification", in *proceedings 9th International Symposium on Methodologies for Intelligent Systems ISMIS'96*, Zakopane, Poland, June 1996.
8. Coenen, F. and Bench-Capon, T., "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.
9. Debenham, J.K., "A Framework For Knowledge Reuse" in *proceedings 11th International FLAIRS Conference*, Florida, May 1998, pp199-203.
10. Debenham, J.K. , "*Knowledge Engineering*", Springer-Verlag, 1998.
11. Walker, A., Kowalski, R., Lenat, D., Soloway, E. and Stonebraker, M., "Knowledge Management", in (L. Kerschberg, Ed.), "*Proceedings from the Second International Conference on Expert Database Systems*", Benjamin Cummings, 1989.
12 Ito, H., "Interface for Integrating a Knowledge-Based System and Database Management System Using Frame-Based Knowledge Representation", in *proceedings Expert Systems World Congress*, J. Liebowitz (Ed), Pergamon Press 1991.
13. Date, C.J., "*An Introduction to Database Systems*" (4th edition) Addison-Wesley, 1986.