Modeling Method Ontologies: A Foundation for Enterprise Model Integration

Christopher Menzel

Texas A&M University
College Station, TX 77843-4237
cmenzel@tamu.edu

Abstract

The importance — or, at least, perceived importance — of enterprise modeling is attested to not only by the sheer numbers of enterprise models but also by the wide variety of modeling methods. This proliferation of modeling methods is a double-edged sword. On the one hand, methods tailored to specific kinds of information enable modelers to create clear, better focused models of the desired sort. On the other hand, because most enterprise modeling methods are ill-defined, both legacy models and current models are difficult to interpret by anyone other than the original creators. Hence, both the reuse of legacy models and the integration of current models across different aspects of an enterprise are, for all practical purposes, impossible. The purpose of this paper is to illustrate an approach to the definition of modeling method ontologies that provides a rigorous foundation for the reuse and integration of enterprise models.

Introduction

The importance — or, at least, perceived importance — of enterprise modeling is attested to not only by the sheer numbers of enterprise models but also by the wide variety of modeling methods: there are methods for function modeling, database modeling, conceptual schema modeling, process modeling, object-oriented design modeling, project plan modeling, and so on. This proliferation of modeling methods is a double-edged sword. On the one hand, there are many different types of information that need to be modeled in a large enterprise — the relatively static information in a database schema, for example, differs considerably in character from the dynamic information involved in a project plan or a manufacturing process. By designing a modeling method to represent a specific type of information, features of situations to be modeled that are extraneous (relative to that type) are filtered out, and relevant features brought to the fore. One is thereby able to create clear, better focused models of the desired sort. For this reason, method proliferation is good.

On the other hand, proliferation has limited both the reusability of legacy models on subsequent projects and the degree to which current models can be integrated with one another. The root of these problems is that most enterprise modeling methods are ill-defined (if defined at all): the languages that are used are not defined in terms of proper grammatical rules, and the intended meanings of the constructs of the language are often presented informally in a manner that leaves even basic issues of interpretation unclear. Consequently, both legacy models and current models are difficult to interpret by anyone other than the original creators. Hence, both the reuse of legacy models and the integration of current models across different aspects of an enterprise are, for all practical purposes, impossible (Menzel *et al.* 1991).

The problem is very much a problem of ontology (Gruber 1991): in adopting a given modeling method, a modeler commits to a distinctive specialized terminology with its own logic — the ontology the method uses to structure the specific type of domain information for which it has been designed. Hence, to be able to reuse a given model, or integrate it with other models, the underlying method ontology must be made explicit and precise.

It follows that a central task that must be undertaken to achieve enterprise model integration, and hence a reasonable degree of enterprise integration generally, is the definition of rigorous ontologies for all widely used enterprise modeling methods. The purpose of this paper is to illustrate by example an approach to this task. Specifically, I will first introduce a simple, informal database schema modeling method called I1X. I will then make its ontology explicit in the form of a first order theory (called FI1X) with an explicit formal semantics. I will then define the notion of an I1X enterprise model in terms of this theory. I will close by suggesting that this approach can be extended to a general theoretical account of enterprise model integration.

I1X: A Simple Data Modeling Method

Informal Overview

I1X is a simplified version of the data modeling language IDEF1X (Brown 1993), a framework for designing relational database schemas. As in IDEF1X, the three most prominent classes of things in our I1X ontology are *entity*

types,[1] *attributes*, and *links*, or *relationships*. To spell out the natures of these basic classes, three further classes need to be admitted: *individuals*, *attribute value domains*, and *attribute sets*. Entity types are, roughly, classes of individuals. (There are a number of different ways of clarifying this further — part of the task of a formal semantics such as the one introduced below is to do so explicitly.) Individuals are thus, by definition, the sort of thing that can be an instance of an entity type. The concept of an individual is thus understood broadly to include both concrete things like employees and timesheets, or more abstract objects like company policies and stock prices.

Attributes are simply functions on individuals, typically the instances of a given entity type. To each attribute is associated a specific attribute value domain, which is the set of possible values that a given attribute can take when applied to an individual. Thus, the value domain of the attribute might be specified to be, say, the set of triples $\langle s,s',s''\rangle$, where s, s', and s'' are finite strings of letters, while the value domain of might be a number under, say, 10,000,000, representing the salary of an employee in US dollars. (In an actual populated relational database, an individual instance of an entity type is represented by a tuple $\langle v_1,\ldots,v_n\rangle$ consisting of all the attribute values assigned to that individual by the attributes associated with that entity type.) Attribute sets are just that: sets of attributes. These are needed to talk about certain privileged sets of attributes — *key classes* — that are associated with entity types (relative to any given schema). Basically, a key class for an entity type e is a set of attributes of e that jointly distinguish every member of the type from every other; more exactly, c is a key class for e if and only if, for each pair of distinct instances x and y of e there is some attribute a in c such that the value of a on x is different from the value of a on y. The I1X *key class condition* states that every entity type must have an associated key class.

Finally, links are general relationships between instances of two entity types. Thus, the link works_for between the employee entity type and the department type would relate each employee to the department he or she works for. Like attributes, links can be thought of *functions*. That is to say, a link connects each instance in its domain — the *child*, or *subordinate*, entity type in the link — and maps it to exactly one instance of its range — the *parent*, or *superordinate*, entity type in the link. Thus, in the above example, employee is the child entity of the works_for link, and department the parent. Links come in three flavors, or *cardinalities*, in I1X: *strong many-to-one*, *one-to-one*, and *nonspecific*. In a strong many-to-one link, each instance of the parent type is related to at least one instance of the child type *via* that link. Thus, typically, works_for between employee and department would be a strong many-to-one link, as every department has at least one employee working for it. A one-to-one link, by contrast, indicates that for any instance of the parent type there is no more than one instance of the child type mapped to it via that link. A nonspecific link puts no constraints on the connection between the child type and the parent type (beyond the mere requirement that the link be functional, i.e., that every instance of the child type is related to exactly one instance of the parent type).

These six classes, then, will be taken as primitive in our simplified data modeling framework I1X.

Entity types, attributes, and links are assembled together into *schemas*, i.e., roughly, collections of entity types with attributes and key classes that are connected together by links. Schemas are represented by I1X *models*, which are typically depicted in a graphical language in which labeled boxes represent entity types — associated attributes and key classes being named within the boxes — and labeled lines of a certain sort represent links. To illustrate, consider, the following little I1X model, which represents a schema involving three entity classes, emp, dept, and div ("division").

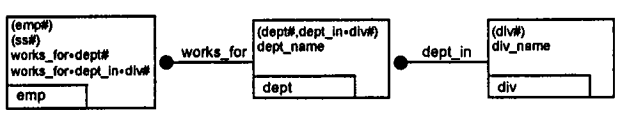

Figure 1. An I1X model

The line between emp and dept labeled "works_for" indicates that every employee works for some (one) department, and, similarly, the line between dept and div labeled "dept_in" indicates that every department is a department in some (one) division. The filled in circle at the "child" end of each line indicates that the corresponding links are strong many-to-one.

Key classes are represented by the lists of attribute names in parentheses. Thus, the fact that both "emp#" and "ss#" are parenthesized indicates that every employee has both a unique employee number and a unique social security number. The compound expressions containing the dot "•" indicate *derived* attributes.[2] To clarify, note

[1] Entity types are often, rather unfortunately, called simply "entities". This is unfortunate as a common meaning of the term "entity" is something like "object" or individual", thus inviting possible confusion of entity types with the objects that instantiate them.

[2] The usual IDEF1X practice is to suppress the link names in derived attribute names when there is no danger of ambiguity; thus, for instance, works_for•dept# in employee would typically be written simply as "dept#". If there were more than one link from employee to division, of course, then the compound attribute names would have to be given in full (or at least annotated appropriately to reflect this), as dept# and div# would both be inherited with respect to both links.

that one can assign the department number of an employee's department to the employee x in the following manner: (i) trace up the works_for link to find x's dept, (ii) determine y's dept#, (iii) assign that value to x. The name "works_for•dept#" indicates the attribute that is derived by this procedure (hence the moniker "derived attribute"), and the attribute dept# is said to be *inherited* by emp (via the link works_for). As the attribute works_for•dept_in•div# shows, a derived attribute (viz., dept_in•div# in this case) can itself be inherited. The I1X *inheritance condition* determines exactly which attributes are to be inherited: every attribute occurring in a key class of the parent entity type of a link is to be inherited by the child type.

Formalization

This, then, is the simple modeling method we want to formalize as a first-order theory. Formalization proceeds in four steps: (i) define the language of the theory; (ii) define the semantics for the language; (iii) axiomatize the semantics in a "foundational theory;" and (iv) define the notion of a model in the theory formally. We use a standard first-order framework (cf., e.g., Mendelson 1987).

The Language L of I1X. The basic language L of I1X is a first-order language with identity but without individual constants. For brevity's sake, I will use a standard first-order language rather than KIF (Genesereth and Fikes 1992), even though, in practice, KIF is the more likely format for the implementation of any of these ideas. Specifically, then, L contains individual variables, the boolean operators $\wedge$, $\vee$, and $\sim$, the quantifier symbols $\forall$ and $\exists$, the identity predicate =, the predicates listed below, and a single 2-place term function symbol •.

The basic 1-place predicates of are:

- Ind
- ET
- L
- Att
- AS
- AVD
- Sch

Intuitively, these predicates pick out the basic classes in the ontology of I1X. The formulas 'Ind(x)', 'ET(x)', 'L(x)', 'Att(x)', 'AS(x)', 'AVD(x)', and 'Sch(x)' can be read as "x is an individual", "x is an entity type", "x is a link", "x is an attribute", "x is an attribute set", "x is an attribute value domain", and "x is a schema", respectively. In addition to these predicates we have the following *n*-place predicates (along with their intended readings, which also indicate their intended arity).

- ET-in ("x is an entity type in y")
- ATS ("x is an attribute in y relative to z")
- KC ("x is a key class of y relative to z")
- Links ("x links y to z in w")
- Inst ("x is an instance of y in z")
- App ("x applied to y has value z")
- Map (x maps y to z")
- In ("x is in (the set) y")

The grammar of L is as follows.

1. Every variable is a *term of L*.
2. If τ and τ' are terms of L, so is $\tau \bullet \tau'$.
3. If π is an *n*-place predicate and $\tau_1, \ldots, \tau_n$ are terms, then $\pi\tau_1 \ldots \tau_n$ is a *formula of L*.
4. If φ and ψ are formulas of L and ν a variable, then $\sim\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\exists\nu\varphi$ and $\forall\nu\varphi$ are also formulas of L.
5. Nothing else is a term or formula of L but those expressions generated by 1-4.

Basic Semantics for I1X. Note again that the suggested readings for the I1X predicates are for heuristic purposes only. We fix their meanings rigorously (and, in this paper, rather simply) by means of the following semantic theory. The framework for expressing the semantics of I1X, i.e., the *metalanguage L** for I1X, is first-order set theory augmented with names and predicates that enable it to talk about the elements of L and define certain semantic notions like that of an interpretation for L, and of the truth of an I1X model in an interpretation. In particular, L* will contain names for each predicate of L, formed by placing each predicate inside single quotes, as well as metavariables that range over certain classes of expression. For the sake of readability, we will not use L* proper; rather, we will use a version of "mathematical English" that incorporates L*. But note that this is for readability only; what we are giving is a formal semantic theory of I1X, which is just itself another, specialized first-order theory.

An *interpretation* I of the language L of I1X is a pair $\langle D, val \rangle$. To define D, first let $D^* = \bigcup \{\iota, \varepsilon, \lambda, \alpha, \sigma\} \cup V$, where

- V is a nonempty set of sets
- ι is a nonempty set
- $\varepsilon = \mathrm{Pow}(\iota)$[3]
- $\lambda = \{f \mid f : e \longrightarrow e', \text{ for } e, e' \in \varepsilon\}$
- $\alpha = \{a \mid a : e \longrightarrow v, \text{ for } e \in \varepsilon \text{ and } v \in V\}$.

Then $D = D^* \cup \mathrm{Pow}^{<\omega^+}(\alpha)$, where $\mathrm{Pow}^{<\omega^+}(\alpha)$ is the set of all nonempty finite subsets of α. Intuitively, ι represents a class of individuals (ordinary objects, data objects, whatever). For the illustrative purpose of this simple semantics, entity types are taken to be extensional entities, i.e., sets. Thus, we define the class ε of entity types (over

[3] Where Pow(D) is the "power set" of D, i.e., the set of all of D's subsets.

ι) simply to be the class of all sets over ι. Accordingly, we take the class of all possible links between entity types to be the set λ of all functions from entity types to entity types, i.e., from members of ε to members of ε. Similarly, the class of all possible attributes on instances of entity types is defined as the set α of all functions from entity types to attribute value domains, i.e., members of V.

The class σ represents the class of all *schemas*. To define the notion of a schema it is easiest first to define the auxiliary notion of a *proto-schema*. A proto-schema is a triple $\langle\varepsilon',\lambda',\alpha'\rangle$, where ε' is a subset of ε (representing some selected subset of entity types), λ' is a subset of the set of functions from members of ε' to members of ε' and α' a subset of the set of functions from members of ε' to sets of attribute values (representing attributes of the entity types in ε'); formally:

DEF: $\langle\varepsilon',\lambda',\alpha'\rangle$ is a *proto-schema of* I iff $\varepsilon' \subseteq \varepsilon$, $\lambda' \subseteq \{f \in \lambda : \mathrm{dom}(f) \in \varepsilon' \text{ and } \mathrm{range}(f) \in \varepsilon'\}$, and $\alpha' \subseteq \{a \in \alpha : \mathrm{dom}(a) \in \varepsilon'\}$.

If $\Sigma = \langle\varepsilon',\lambda',\alpha'\rangle$ is a proto-schema of I, let $\varepsilon_\Sigma = \varepsilon'$, $\lambda_\Sigma = \lambda'$, and $\alpha_\Sigma = \alpha'$. These notational conventions will enhance the readability of the definitions to follow. The notion of a proto-schema is useful for defining that of a full-fledged schema, as a schema will just be a proto-schema that satisfies a couple additional conditions.

So let Σ be a proto-schema. Then a finite subset α'' of α_Σ is said to be a *key class* for e in Σ just in case α'' distinguishes every member of e from every other member; more exactly:

DEF: Let Σ be a proto-schema, and let $e \in \varepsilon_\Sigma$. Then $\alpha'' \subseteq \alpha_\Sigma$ is a *key class for e in* Σ iff α'' is finite, $\mathrm{dom}(a) = e$ for all $a \in \alpha''$, and for every $x,y \in e$, if $x \neq y$, then there is an $a \in \alpha''$ such that $a(x) \neq a(y)$. e is said to *have a key class in* Σ iff some $\alpha'' \subseteq \alpha_\Sigma$ is a key class for e in Σ.

DEF: A proto-schema Σ *satisfies the key class condition* iff every $e \in \varepsilon_\Sigma$ has a key class in Σ.

The set $\mathrm{Pow}^{<\omega^+}(\alpha)$ is included in D to ensure that all possible key classes are in the domain.

Our next task is to formalize the inheritance condition. It was noted informally regarding the example above that the attribute dept#•works_for derived from the attribute dept# and the link works_for. What this means formally, given our interpretation of attributes and links as functions, is that the derived attribute dept#•works_for is the *functional composition* of works_for with dept#. dept# is then said to be *inherited* by emp via works_for. More generally, then, an attribute a of e' in Σ is *inherited* by e via a link f in Σ just in case f links e to e' in Σ and the composition $f \circ a$ of f and a is an attribute of e in Σ (where $f \circ a(x) = a(f(x))$); formally,

DEF: Let Σ be a proto-schema, let $f \in \lambda_\Sigma$ be such that $f : e \longrightarrow e'$, and let $a \in \alpha_\Sigma$ be such that $\mathrm{dom}(a) = e'$. Then a is *inherited by e via f in* Σ just in case $f \circ a \in \alpha_\Sigma$.[4]

We then say that Σ *satisfies the inheritance condition* just in case, whenever there is a link f from e to e' in Σ, every attribute in every key class of e' is inherited by e; formally,

DEF: A proto-schema Σ *satisfies the inheritance condition* iff, whenever $f : e \longrightarrow e'$ for $f \in \lambda_\Sigma$, for every key class α'' for e' in Σ, each $a \in \alpha''$ is inherited by e via f in Σ.

Given this, we can define the notion of a schema:

DEF: A proto-schema Σ of I is a *schema of* I iff it satisfies both the key-class condition and the inheritance condition.

Notice that we do not require that a schema be *connected*, in the sense that there is a way of "following links" backwards or forwards from any entity type in a schema to any other — though this common constraint on IDEF1X schemas could easily be defined with our apparatus and added as a further condition. The ease with which such conditions can be defined and added, and the clarity of such definitions, are exactly the sorts of advantages that this example of modeling theory definition is intended to illustrate.

We can now define the element val of an interpretation. Specifically, val is a semantic function that assigns meanings to the elements of L in terms of D as follows.

1. val('Ind') = ι
2. val('ET') = ε
3. val('L') = λ
4. val('Att') = α
5. val('AS') = $\mathrm{Pow}^{<\omega}(\alpha)$
6. val('AVD') = V
7. val('Sch') = σ
8. val('ET-in') = $\{\langle e,\Sigma\rangle \mid \Sigma \in \sigma \text{ and } e \in \varepsilon_\Sigma\}$
9. val('ATS') = $\{\langle a,e,\Sigma\rangle \mid \Sigma \in \sigma, a \in \alpha_\Sigma, e \in \varepsilon_\Sigma, \text{ and } \mathrm{dom}(a) = e\}$
10. val('KC') = $\{\langle \alpha'',e,\Sigma\rangle \mid \Sigma \in \sigma \text{ and } \alpha'' \text{ is a key class for } e \text{ in } \Sigma\}$
11. val('Links') = $\{\langle f,e,e',\Sigma\rangle \mid \Sigma \in \sigma, f \in \lambda_\Sigma, \mathrm{dom}(f) = e \text{ and } \mathrm{range}(f) = e'\}$
12. val('Inst') = $\{\langle x,e\rangle \mid e \in \varepsilon, \text{ and } x \in e\}$
13. val('App') = $\{\langle a,x,v\rangle \mid a \in \alpha, \text{ and } a(x) = v\}$
14. val('Map') = $\{\langle f,x,y\rangle \mid f \in \lambda \text{ and } f(x) = y\}$
15. val('In') = $\{\langle a,\alpha''\rangle \mid \alpha'' \in \mathrm{Pow}^{<\omega^+}(\alpha) \cup V \text{ and } a \in \alpha''\}$

[4] Note that, on this semantics, to say that attribute b is an attribute of an entity type e in S is just to say that it is a member of α and that its domain is e.

The identity predicate, of course, is interpreted as usual. Relative to the notion of entity type at work in this semantics, it should be clear that the interpretations of the predicates all capture their intended meanings.

Finally, we need to interpret terms:

16. If v is a variable, then $val(v) \in D$
17. $val(\tau \bullet \tau') = val(\tau) \circ val(\tau')$, if $range(\tau) = dom(\tau')$; otherwise $val(\tau \bullet \tau') = \varnothing$.

Truth for the formulas of L relative to an interpretation is understood as usual.

The I1X Foundational Theory. Now that we have defined a clear, precise semantics for our language, our goal now is formulate a *theory*, or more evocatively, a *logic*, that captures this semantics axiomatically.

We can distinguish three types of axiom for I1X, "typing" axioms, "substantive" axioms, and "artifactual" axioms. Typing axioms simply stipulate the kinds of things that can stand in the relations indicated by the *n*-place predicates, for $n>1$. Substantive axioms, as the name implies, express somewhat deeper semantic properties. Artifactual axioms express propositions that reflect quirks of the chosen semantics rather than any deep truths about the modeling method's ontology.

First, we define an *attribute value* to be a something that is in an attribute value domain:

$AV(x) =_{df} \exists y(AVD(y) \wedge In(x,y))$

1. If x is an entity type in y, then x is an entity type and y is a schema:

$ET\text{-}in(x,y) \rightarrow (ET(x) \;\&\; Sch(y))$

2. If x is an attribute in y relative to z, then x is an attribute and y is an entity type in z. (Note that it follows from 1. that z is a schema.)

$ATS(x,y,z) \rightarrow (Att(x) \;\&\; ET\text{-}in(y,z))$

3. If x is in y, then either x is an attribute set and y is an attribute, or x is an attribute vale and y is an attribute value domain.

$In(x,y) \rightarrow ((AS(x) \wedge Att(y)) \vee (AV(x) \wedge AVD(y))$

4. If x is a key class in y relative to z, then y is an entity type in z (and hence, by 1., z is a schema), and the only things in x are attributes in y relative to z.

$KC(x,y,z) \rightarrow ET\text{-}in(y,z) \wedge \forall w(In(w,x) \rightarrow ATS(w,y,z))$

5. If x applied to y has value z, then x is an attribute, y is an individual, and z is an attribute value.

$App(x,y,z) \rightarrow (Att(x) \wedge Ind(y) \wedge AV(z))$

6. If x maps y to z, then x is a link and y and z are individuals.

$Map(x,y,z) \rightarrow (Ind(x) \wedge Ind(y) \wedge L(z))$

7. If x links y to z in w, then x is a link, w is a schema, and y and z are entity types in w.

$Links(x,y,z,w) \rightarrow (L(x) \;\&\; ET\text{-}in(y,w) \;\&\; ET\text{-}in(z,w))$

8. If x links y to z in a schema w, then for every instance u of y there is an instance v of z to which x maps u.

$Links(x,y,z,w) \rightarrow \forall u(Inst(u,y) \rightarrow \exists v(Inst(v,z) \wedge Map(x,u,v))$

We need an axiom to govern the meaning of terms formed from the dot operator. The following does the job.

9. If x•y is an attribute in entity type z relative to w, then there is an entity type u such that x links z to u in w, and there is a key class v of u such that y is in v relative to w. (Note that it follows from this and the above that x is a link and y is an attribute.)

$ATS(x \bullet y,z,w) \rightarrow \exists u \exists v(Links(x,z,u,w) \wedge KC(v,u,w) \wedge In(y,v)$

As noted, the *substantive* axioms which follow express the deeper conditions placed on the semantics.

10. Individuals, entity types, links, attributes, attribute sets, attribute value domains, and schemas are all distinct kinds of things.

$(Ind(x) \rightarrow \sim(ET(x) \vee (L(x) \vee Att(x) \vee AS(x) \vee AV(x) \vee AVD(x) \vee S(x))) \wedge (ET(x) \rightarrow \sim(L(x) \vee Att(x) \vee AS(x) \vee AV(x) \vee AVD(x) \vee S(x))) \wedge (L(x) \rightarrow \sim(Att(x) \vee AS(x) \vee AV(x) \vee AVD(x) \vee S(x))) \wedge (Att(x) \rightarrow \sim(AS(x) \vee AV(x) \vee AVD(x) \vee S(x))) \wedge (AS(x) \rightarrow \sim(AV(x) \vee AVD(x) \vee S(x))) \wedge (AV(x) \rightarrow \sim(AVD(x) \vee S(x))) \wedge (AVD(x) \rightarrow \sim S(x))$

11. Every link in a schema links exactly one entity type to exactly one other (possibly the same) entity type.

$(Links(x,y,z,w) \wedge Links(x,y',z',w)) \rightarrow (y = y' \wedge z = z'))$

12. An attribute can be an attribute in only one entity type relative to a given schema.

$(ATS(x,y,z) \wedge ATS(x,y',z)) \rightarrow y = y'$

The next axiom connects the entity type that an attribute is in with the object to which it applies.

13. If an attribute x is an attribute in entity type y in schema z, then x has a value when applied to every instance of y.

$ATS(x,y,z) \rightarrow \forall w(Inst(w,y) \rightarrow \exists u App(x,w,u))$

14. If z and z′ are both values of an attribute x applied to an individual y, then z=z′.

$(\mathsf{App}(x,y,z) \wedge \mathsf{App}(x,y,z')) \rightarrow z = z'$

An analogue of the above axiom exists for the predicate 'Map'.

15. If x maps y to both z and z', then z = z'.

$(\mathsf{Map}(x,y,z) \wedge \mathsf{Map}(x,y,z') \rightarrow z = z'$

In virtue of Axiom 14 and Axiom 15, we adopt the following notational abbreviation:

DEF: $x(y) = z \equiv_{df} \mathsf{App}(x,y,z) \vee \mathsf{Map}(x,y,z)$

The typing axioms 5 and 6 together with Axiom 10 will determine which of the two disjunctions is the relevant one in any given context. When we know that Att(x) or L(x), we will often write 'x(u)' to indicate the unique z such that x(u) = z.

Next we introduce two definitions that enable us to express a link's cardinality in a schema.

DEF: $\mathsf{SM1}(x,w) \equiv_{df} \exists y \exists z(\mathsf{L}(x,y,z,w) \wedge \forall u(\mathsf{Inst}(u,z) \rightarrow \exists v(\mathsf{Inst}(v,y) \wedge x(y) = u)))$

That is, x is *strong many-to-one in* w iff x links some y to some z in w, and every instance of z has an instance of y mapped onto it by x.

DEF: $\mathsf{1to1}(x,w) \equiv_{df} \exists y \exists z(\mathsf{L}(x,y,z,w) \wedge \forall u(\mathsf{Inst}(u,z) \rightarrow \sim\exists v \exists v'(\mathsf{Inst}(v,y) \wedge \mathsf{Inst}(v',y) \wedge v \neq v' \wedge x(v) = u \wedge x(v') = u)))$

That is, x is *one-to-one in* w iff x links some y to some z in w, and no instance of z has more than one instance of y mapped onto by x.

16. Attribute sets that contain the same attributes are identical.

$(\mathsf{AS}(x) \wedge \mathsf{AS}(x') \wedge \forall w(\mathsf{In}(w,x) \leftrightarrow \mathsf{In}(w,x'))) \rightarrow x = x'$

17. Every key class is an attribute set.

$\mathsf{KC}(x,y,z) \rightarrow \mathsf{AS}(x)$

18. Every attribute set contains at least one thing. (Note that it follows from 3. that that thing is an attribute.)

$\mathsf{AS}(x) \rightarrow \exists w \mathsf{In}(w,x)$

19. If x is a key class of y in z, then every two distinct instances of y differ with respect to at least one of the attributes in x.

$\mathsf{KC}(x,y,z) \rightarrow \forall w \forall w'((\mathsf{Inst}(w,x) \wedge \mathsf{Inst}(w',x) \wedge w \neq w') \rightarrow \exists u(\mathsf{In}(u,x) \wedge u(w) \neq u(w')))$

The following axiom expresses the key class condition.

20. Every entity type in a schema has a key class relative to that schema.

$\mathsf{ET\text{-}in}(x,z) \rightarrow \exists y \mathsf{KC}(y,x,z)$

It will be convenient to introduce a piece of defined notation. Let x be a variable distinct from the terms $t_1,\ldots,t_n$ which does not occur in φ.

DEF: $\varphi([t_1,\ldots,t_n]) \equiv_{df} \exists x(\mathsf{AS}(x) \wedge \varphi(x) \wedge \forall y(\mathsf{In}(y,x) \leftrightarrow (y = t_1 \vee \ldots \vee y = t_n)))$

To see what is going on here, note that, by Axiom 3, the only values of x that can make the biconditional $\forall y(\mathsf{In}(y,x) \leftrightarrow (y = x_1 \vee \ldots \vee y = x_n))$ in the right side of the definition true for any values of the x_i are finite sets of attributes (i.e., members of $\mathrm{Pow}^{<\omega^+}(\alpha)$); they are the only things that other things (attributes, as it happens) can bear the *In* relation to. Consequently, what the right side of the definition says, in effect, is that the formula φ is true of some set of attributes whose members are exactly (denoted by) $t_1,\ldots,t_n$. By Axiom 16, this set is unique. Hence, we are warranted in forming a name for that set out of names for its members. In the special cases where $\varphi(x)$ is a formula of the form $\mathsf{KC}(x,t,t')$, for any terms t, t', we call $\varphi([t_1,\ldots,t_n])$ a *key class formula.*

The following axiom expresses the inheritance condition.

21. If x links y to z in w, then every attribute in every key class of z is inherited by y via x.

$\mathsf{Links}(x,y,z,w) \rightarrow \forall u(\mathsf{KC}(u,z,w) \rightarrow \forall v(\mathsf{In}(v,u) \rightarrow \mathsf{ATS}(x \bullet v,y,w)))$

Nearly any mathematical semantics for a first-order theory — especially applied theories like modeling theories — is going to have certain features that may not reflect essential features of the intuitive realm one is attempting to capture. However, these features may be expressible in one's language, and hence, despite their nonessential character, end up being logical truths relative to the chosen semantics. For example, on our chosen semantics, in virtue of our use of sets to model entity types, we have that

22. Entity types with the same instances are identical.

$(\mathsf{ET}(x) \wedge \mathsf{ET}(y)) \rightarrow (\forall w(\mathsf{Inst}(w,x) \leftrightarrow \mathsf{Inst}(w,y)) \rightarrow x = y))$

Indeed, arguably, this is more than a mere artifact, but a reflection of an inadequate semantical treatment of entity types; entity types just aren't sets, since they can change their instances over time (whereas a set has exactly the instances it has, i.e., exactly the members it has, essentially; change the membership and you have a new set. Our formal semantics thus foists a feature onto entity types that is decidedly *not* a part of the intuitive semantics of the theory.

Along the same lines, the following is a logical truth of our theory:

23. Attributes that agree on the values they assign to all objects are identical.

$(\mathsf{Att}(x) \wedge \mathsf{Att}(y)) \rightarrow (\forall w \forall z(\mathsf{App}(x,w,z) \leftrightarrow \mathsf{App}(y,w,z)) \rightarrow x=y))$

Similarly for links:

24. Links that map the same individuals to the same individuals are identical.

$(\mathsf{L}(x) \wedge \mathsf{L}(x') \wedge \forall y \forall z(\mathsf{Map}(x,y,z) \leftrightarrow \mathsf{Map}(x',y,z))) \rightarrow x = x'$

The extensionality of attributes has related implications for the key classes of an entity type.

25. If $x_1, \ldots, x_n$ are all attributes of an entity type y (relative to schema w) such that every instance of y is distinguished from every other instance by at least one of those attributes, then there is a key class for y relative to w consisting of exactly $x_1, \ldots, x_n$.

$(\mathsf{ATS}(x_1,y,w) \wedge \ldots \wedge \mathsf{ATS}(x_n,y,w) \wedge \forall z \forall z'((\mathsf{Inst}(z,y) \wedge \mathsf{Inst}(z',y) \wedge z \neq z') \rightarrow (x_1(z) \neq x_1(z') \vee \ldots \vee x_n(z) \neq x_n(z')))) \rightarrow \mathsf{KC}([x_1,\ldots,x_n],y,w)$

Intuitively, however, it should not be enough for a set of attributes to constitute a key class for a given entity type that it just *happen* to uniquely individuate the instances of the type, as it might fail to do so relative to another set of instances. Intuitively, that is, a key class for a type must *necessarily* individuate the instances of the type. However, as Axiom 22 reflects, the semantics for I1X at hand does not differentiate between a type and the set of its instances. Hence, the semantics cannot represent the idea of a single entity type having different possible sets of instances — nor is the current language capable of expressing it; for that we will need go to a *modal* language. There is thus a trade-off in the formalization of modeling theories: a simple semantics tends to generate a greater number of artifactual truths. One must decide on a case-by-case basis whether the gain in simplicity is worth the cost.

Soundness

We note that these axioms are all true as far as they go. We express this in a *soundness* theorem. Call the set of twenty-five axioms above the *foundational theory for I1X*, or F_{I1X} for short.

THEOREM (Soundness): *The foundational theory F_{I1X} for I1X is sound. That is, every axiom of the theory is true in every interpretation.*

At this stage it is not clear whether these axioms as they stand are complete, that is, whether every sentence that is true in all interpretations is derivable from these axioms. However, because they are expressively quite weak (there is no underlying notion of number, and only the thinnest notion of set is needed for key classes), there seems nothing about them that would indicate that they are *essentially* incomplete. Thus, the issue is simply whether any further axioms are needed for completeness, or whether the axioms above suffice. If the former, any axioms that might be missing should be revealed in the course of constructing a completeness theorem.

Theorems

If the axioms for a modeling theory are complete or, at least, reasonably comprehensive, it should be possible to prove some substantive theorems. For instance, in the case of F_{I1X}, we can prove the following.

THEOREM : *If x links y to z in w and x is a one-to-one link, then the collection of attributes inherited by y from any key class of z generates a corresponding key class for y.*

In symbols:

$(\mathsf{Links}(x,y,z,w) \wedge \mathsf{1to1}(x)) \rightarrow \forall u(\mathsf{KC}(u,z,w) \rightarrow \exists u'(\mathsf{KC}(u',y,w) \wedge \forall v((\mathsf{In\text{-}KC}(v,u,w) \rightarrow \mathsf{In\text{-}KC}(x \bullet v,u',w)) \wedge (\mathsf{In}(v,u') \rightarrow \exists v'(\mathsf{In}(v',u) \wedge v = x \bullet v')))))$

Proof. We will prove this in logical English rather than as a formal derivation. Suppose the antecedent is true, i.e., that a one-to-one link r links e1 to e2 in s. Let k be a key class for e2. By Axiom 21, if a is in k (i.e., if In(a,k)), then r•a is an attribute of e1, for any attribute a. So let k′ be the set of all attributes of the form r•a where In(a,k), and let x and y be distinct instances of e1 in s. Since r is one-to-one, it follows by definition that $r(x) \neq r(y)$. Since k is a key class for e2, it follows that there is an attribute a in k such that $a(r(x)) \neq a(r(y))$, i.e., $r \bullet a(x) \neq r \bullet a(y)$. Hence, it follows by Axioms 19 and 20 that there is an attribute a′ in k′ (viz., r•a) that distinguishes x and y, i.e,. an such that a′ such that $a'(x) \neq a'(y)$. Since x and y were chosen arbitrarily, the same is true for any pair of distinct instances of e1, and so, by definition, k′ is a key class for e1.

I1X Models

We turn now to the definition of the notion of an I1X *model*. An I1X model is simply a first-order theory that extends the foundational theory in a certain way. Specifically, first, we extend L to a new language L^+ by adding a finite number of constants $c_1, \ldots, c_m$ to L. (Intuitively, these will serve as names for particular entity types, attributes, and links.) Call such an extension an *enrichment of* L. Given this, we define an I1X model as follows.

DEF: Let L^+ be an enrichment of L, and let s be a specific constant of L^+. An *I1X model set* M is any set of sentences of L^+ consisting of

- A single schema axiom Sch(s).
- A finite number of entity type axioms ET-in(t,s), t any constant of L^+.
- For each t such that ET-in(t,s) $\in$ M, a finite number of attribute axioms ATS(t',t,s), t' any constant of L^+.
- For each t such that ET-in(t',s) $\in$ M, a finite number of key class axioms KC([t_1,...,t_m],t,s), t_1, ..., t_m any terms of L^+.
- A finite number of link axioms Links(t,t',t'',s), t any constant of L^+, t', t'' such that ET-in(t',s) $\in$ M and ET-in(t'',s) $\in$ M.
- For each t such that Links(t,t',t'',s) $\in$ M, at most one cardinality axiom SM1(t) or 1to1(t).

DEF: An *I1X model* is a first-order theory $F_{I1X} \cup M$, where M is any I1X model set.

DEF: An *interpretation* $I^+ = \langle D,V,val^+ \rangle$ for an enrichment L^+ is the result of extending the valuation function val in an interpretation $I = \langle D,V,val \rangle$ for L such that val maps each new constant of L^+ to an object in D.

Given this, we say that an I1X model $M^* = F_{I1X} \cup M$ *holds* in an interpretation I^+ if and only if every sentence in M is true in I^+.

As an example, consider our simple I1X model once again (call it "A1").

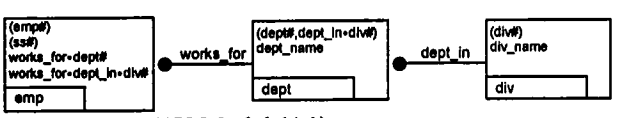

Figure 1: An I1X Model (A1)

To capture this model as a first-order theory, we first enrich L with the constants s, emp, dept, div, emp#, ss#, dept#, div#, dept_name, div_name, works_for, and dept_in. We then construct the following model set M1:

Schema axiom: Sch(s)

Entity type axioms: ET-in(emp,s), ET-in(dept,s), ET-in(div,s)

Attribute axioms: ATS(emp#,emp,s), ATS(ss#,emp,s), ATS(dept#,dept,s), ATS(dept_name,dept,s), ATS(div#,div,s), ATS(div_name,div,s)

Key class axioms: KC([emp#],emp,s), KC([ss#],emp,s), KC([dept#,dept_in•div#],dept,s), KC([div#],div,s)

Link axioms: Links(works_for,emp,dept,s), Links(dept_in,dept,div,s)

Cardinality axioms: SM1(works_for,s), SM1(dept_in,s)

The theory $M1^* = F_{I1X} \cup M1$, then, constitutes the first-order theory corresponding to the graphical model above.

Note that, theoretically speaking, there is some redundancy in M1, as, e.g., KC([emp#],emp,s) implies both ET-in(emp,s) and ATS(emp#,emp,s) (as well as other sentences not included in M1 such as ATS(works_for•dept#,emp,s)), etc., since these sentences are derivable from M1*. However, adding the redundant axioms makes the correlation between the graphical model and the model set more explicit, and so is heuristically useful.

There is no guarantee that an I1X model so defined makes any sense. In particular, it is possible to put statements in a model set that are inconsistent with the foundational theory. For example, one might include both the statement ET-in(e,s) and the statement ATS(e,e',s) in a model set. The latter, by Axiom 2 entails Att(e), while the former, by Axiom *1.* entails ET(e). However, Att(e) and ET(e) are jointly inconsistent with Axiom 10. Thus, we add the following definition.

DEF: An I1X model $M^* = M \cup F_{I1X}$ in the enrichment L^+ of L is *coherent* if and only if M* holds in some interpretation of L^+.

Basic Model Integration

Now, due to its simplicity there are certain limitations of this formal framework, especially with regard to the integration of models. However, it is useful enough to illustrate certain kinds of model integration. Note that I will ignore here the separate but crucially important practical problem of how, in practice, to translate the content of two models into the same language.

Consider, then, another simple I1X model, A2.

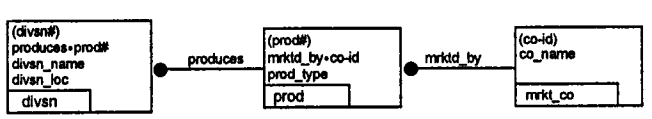

Figure 2: I1X Model A2

Obviously, a first-order model set M2 for A2 can be written out in the same fashion as the model set M1 of A1. We will suppose this to have been done. In particular, there will be an entity type axiom ET-in(divsn,s2), where 's2' is the constant chosen to name the schema expressed by A2. (If we want to talk about the schemas expressed by A1 and A2, then, of course, we need to use different names for them.)

Suppose that models A1 and A2 are models within the same enterprise. Clearly, there is reason to suspect that div and divsn are the same entity type. Actually determining whether this is so, of course, is nothing that logic of itself can tell us (though there are of course heuristics that can

aid in so determining). One this *has* been determined, however, that fact can be represented explicitly. The procedure is this. First, we add both model sets M1 and M2 to the foundational theory F_{IIX}. Call this theory IM (= $F_{IIX} \cup M1 \cup M2$). This gives us the basis for an *integration theory* of the two models. Note that there are no confusions about what fact came from what model set, as all the facts that are schema specific (e.g., that ss# is an attribute in emp) are appropriately indexed. To establish the identity of div and divsn, then, we simply add the axiom div = divsn to IM. By the laws of identity, all properties ascribed to divsn in s2 will apply to div.

As things stand, this may not seem all that interesting. For most all of the interesting properties ascribed to divsn, i.e., equivalently, to div, are relative to either s or s2. For example, from div = divsn one can infer ATS(produces•prod#,div,s2) from the M2 model set axiom ATS(produces•prod#,divsn,s2) by the usual laws of identity, though it is difficult to see much use for such information for the users of either model. However, the importance of model integration stems from the fact that there are *logical connections*, or *constraints*, between two or more models. Often the constraints that connect two models are only implicit, existing in virtue of the nature of the information in the models but not explicitly identified. In an integrated environment, such constraints are explicitly identified and maintained.

To illustrate, suppose that div and divsn are not identical, but that, to reflect contextual usage in the enterprise in which the models were created, div_name always yields a value of the form <string1>-<string2>, where <string1> is some sort of descriptive name and <string2> is the name of the division location given by div_loc, and that divsn_name yields only the descriptive name. There is thus an implicit constraint between the two models, viz.,

The div_name of any instance x of div consists of the value of divsn_name on x (assumed to be a string) followed by a hyphen followed by the value of divsn_loc on x.

More formally, this might be expressed as follows:

$$\forall x(\mathrm{Inst}(x,\mathrm{div}) \rightarrow \mathrm{div\_name}(x) = \mathrm{divsn\_name}(x)\,\hat{}\,\text{'-'}\,\hat{}\,\mathrm{divsn\_loc}(x)),$$

where ^ is an operator indicating string concatenation and '-' a name for the hyphen. (The concatenation operator and quotation would have to be part of a suitable axiomatization of strings in a complete treatment, of course.) By adding this axiom to the integration theory IM, the implicit constraint is now explicit, and, to this extent, the models have been logically integrated.

This suggests the definition of one type of integration.

DEF: Two models are *logically integrated* iff all relevant constraints between the models have been identified and made explicit as first-order axioms in an integration theory for the two models.

It is, of course, one thing for constraints to be identified in some environment, and quite another for them to be maintained in some sense; intuitively, the constraint connecting A1 and A2 must be maintained if the models are to remain consistent. The exact meaning of "maintained" for a given constraint in a given environment will vary from case to case. In some cases, it will involve actual changes to a model (for example, when a change in a financial model impacts an evolving product design model), in other cases changes to a database created in accordance with a model. Maintenance of the above constraint would, presumably, be of the latter sort. In a setting in which databases have been structured in accordance with A1 and A2, the constraint might be maintained by implementing a link between div and divsn such that records corresponding to instances of div and divsn would be dynamically linked. Thus, the value of div_name in an record for an instance of div in the database for A1 would not be stored separately, but would be constructed directly from the values of divsn_name and divsn_loc in the corresponding record for that instance in the database for A2. In this way, changes and additions to A2's database would propagate directly to the corresponding database for A1.

This suggests a semi-formal definition of a more practical type of integration:

DEF: Two models are *dynamically integrated* iff they are logically integrated and all identified constraints between them are maintained over time.

Ambiguity

A natural question concerns the treatment of ambiguity across models. In general, since the models essentially represent different contexts, we cannot know a priori whether two uses of the same name in different models refer to the same semantic object, e.g., the same entity type. For example, suppose that, instead of A2, we encountered a slight variant, viz.,

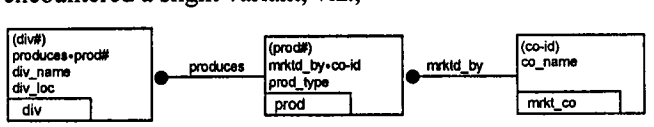

Figure 3. Model A3

If we use constants that are identical with the names in the model, then the first-order model set M3 for A3 would contain, e.g., the entity type axiom ET-in(div,s3), where 's3' names the schema expressed by the model. If we were then to form an integration theory for M1 and M3 by

simply taking the union of M1 and M3 with F_{I1X} without further inquiry, we would in effect be declaring that the div of A1 is the div of A3. Granted, this *might* be true, but it equally might not; 'div' might simply be used ambiguously in the enterprise to mean one thing in one context and something else in another.

Though often a difficult practical problem, the theoretical solution is obvious: initially, anyway, the use of the same name in different model sets is simply disallowed. This can be accomplished in various ways. For instance, one might adopt a naming scheme that indexes constants in a model set to the corresponding model (as needed). Thus, for instance, instead of using div in M1, one might use div/s or div_s, or something of the sort. Then, when one forms an integration theory from M1 and M3, all there will be are the axioms ET-in(div_s,s) and ET-in(div_{s3},s3), which obviously imply nothing about the identity of the (ostensibly) two entity types in the two models. Should one later discover them actually to be identical, then one can simply add the axiom $div_s = div_{s3}$ (or, more tediously, remove subscripts).

Heterogeneous Integration

The example above illustrates what might be called *strong homogeneous* integration, that is, integration between models generated by the same method. Integration between models generated by different methods that are of the same type (e.g., two database modeling methods like IDEF1X and the Entity-Relationship (ER) method) can be called *weak homogeneous* integration. And integration between models of different types can be called *heterogeneous* integration. Typically, of course, in large enterprises, all three types of integration are needed — constraints exist between models created by many different types of methods. However, both weak homogeneous and heterogeneous integration can be approached by expanding on the approach to strong homogeneous integration articulated above. The expanded approach, though, obviously, will involve the formalization of multiple modeling methods instead of just one. It suffices to consider the approach for two methods, as it generalizes directly for more than two.

First, the integration of two informal, graphical models m and m′ generated by distinct methods T and T′ will, of course, require foundational theories F_T and $F_{T'}$ for both methods. As above, the formal models $A^* = A \cup F_T$ and $A'^* = A \cup F_T$ can be defined given model sets A and A′ for m and m′. An integration theory for these models first requires a *foundational integration theory* $I_{T,T'}$ for the foundational theories F_T and $F_{T'}$. Analogous to an integration theory for models, $I_{T,T'}$ will consist in axioms that capture the logical connections, not between models, but between the *ontologies* of the two modeling methods T and T′. For example, the business function modeling method IDEF0 includes the notion of a *concept* which – the name notwithstanding – can for most purposes be identified with the notion of an IDEF1X (hence, I1X) object type. Hence, the integration theory $I_{T,T'}$ will contain an axiom to that (or some similar) effect, e.g.,

$$\forall x(\text{Concept}(x) \leftrightarrow \text{ET}(x))$$

Subtler axioms, of course, would be added to capture more complex logical connections between the two method ontologies. Given this, we can say that an integration theory for the two models A* and A′* will be a theory $I_{T,T'} \cup A^* \cup A'^* \cup C$, where C is a set of axioms expressing logical constraints between the models A* and A′*. Models A* and A′* can be considered logically integrated, then, just in case the elements of C capture all relevant logical constraints. As before, dynamic integration consists in the maintenance of those constraints in the enterprise at issue.

Acknowledgments

I would like to thank the Commonwealth Scientific and Industrial Research Organization (CSIRO), Division of Manufacturing Technology, Melbourne Laboratory for its generous support while much of this paper was written. My thanks especially to Laszlo Nemes for making my visit possible and so very enjoyable, and to the Intelligent Manufacturing Systems group for their comments on an early version of this paper. My thanks also to Michael Grüninger. The ill-formed ideas out of which this paper grew began to take shape after a stimulating day of discussion with him when our paths crossed in Melbourne. Unclarities and errors that remain, of course, are my own fault (though I won't mind if he shoulders some of the blame).

References

Brown, R. 1993. Integration Definition for Information Modeling (IDEF1X), Federal Information Processing Standard (FIPS) 184, National Institute of Standards and Technology, Gaithersburg, MD.

Genesereth, M. R., and Fikes, R. E. eds. 1992. Knowledge Interchange Format, Version 3.0 Reference Manual Technical Report Logic-92-1, Computer Science Department, Stanford University.

Gruber, T. R. 1991. The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases. *Principles of Knowledge Representation and Reasoning:*

Proceedings of the Second International Conference, 601-602. Cambridge, Mass.: Morgan Kaufmann.

Mendelson, E. 1987. *Introduction to Mathematical Logic.* Monterey, California: Wadsworth.

Menzel, C., Mayer, R., and Sanders, L. 1992. Representation, Information Flow, and Model Integration. In C. Petrie ed. *Enterprise Model Integration: Proceedings of the First International Conference*, 131-141. Cambridge, Mass.: MIT Press.