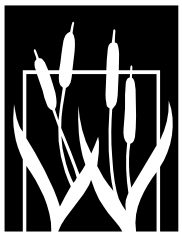

Whitemarsh
Information Systems Corporation

An Olde Saw That Just Won't Cut, A Software Implementation Follow-up

Whitemarsh Information Systems Corporation
2008 Althea Lane
Bowie, Maryland 20716
Tele: 301-249-1142
Email: Whitemarsh@wiscorp.com
Web: www.wiscorp.com

Table of Contents

1. Background

In the TDAN Issue, No 4, March 1998, the article, *An Olde Saw That Just Don't Cut,* presented the "old saw" about data elements, that is, that a data element was composed of three parts: prime word, modifier[s], and a single class word. The TDAN article asserted that this old "don't cut" because of five mistaken notions (i.e., the olde saw's teeth). The five teeth were:

- That data elements are synonyms for table columns, screen cells, entity attributes, or report fields.
- That data elements don't have names in their own right.
- That a data element's name is contrived from a prime word, modifier[s] and class word
- That modifiers are from a single homogeneous set
- That there is only a choice of one class word

This article does not review these five teeth because the original TDAN article can be retrieved from TDAN at http://www.tdan.com/i004fe06.htm. This article has been updated and the revised version is available on the Whitemarsh website. The old TDAN article challenged readers by saying that *This old saw just don't cut it. Never did, and never will. So, stop wasting your time and money and choose one that does.*

In response to that TDAN article, readers and reviewers answered by saying, *Well Whitemarsh, build me a system that solves these problems*. In short, in an adaptation of another old saw, they were telling me to: *put my software were my pen is.* This article reports on that accomplishment, both in terms the of accomplishment and in terms of the five teeth.

While this article is of general interest, it unabashedly presents the Whitemarsh implementation of its metabase component, data modeler. The Whitemarsh metabase embraces the majority of the area identified in the Whitemarsh Knowledge Worker Framework (KWF). A book on the KWF is available for free download from the Whitemarsh website, www.wiscorp.com.

The role of the metadata repository product, metabase, is to capture, retrieve, update, and support the analysis of the metadata necessary for the definition, implementation, and evolution of knowledge worker environments. Information Technology is clearly within the knowledge worker environment, and data modeling is clearly within Information Technology. Hence, within the KWF, data modeling is a critical component.

Within the Whitemarsh data modeler there are six distinct submodels. These are depicted–at a high level of abstraction--in Figure 1. The six submodels are:

- Data Semantics Hierarchies
- Data Elements
- Specified Data Model
- Implemented Data Model
- Operational Data Model
- Application Interface Model

The data semantics model enables the semantic classifications to regularize the rules and meanings of the data represented by the data values associated with the data element, specified, and implemented data models.

The data element submodel contains the metadata necessary to support data element use as semantic templates for the creation of context dependent collections of business facts. That is,

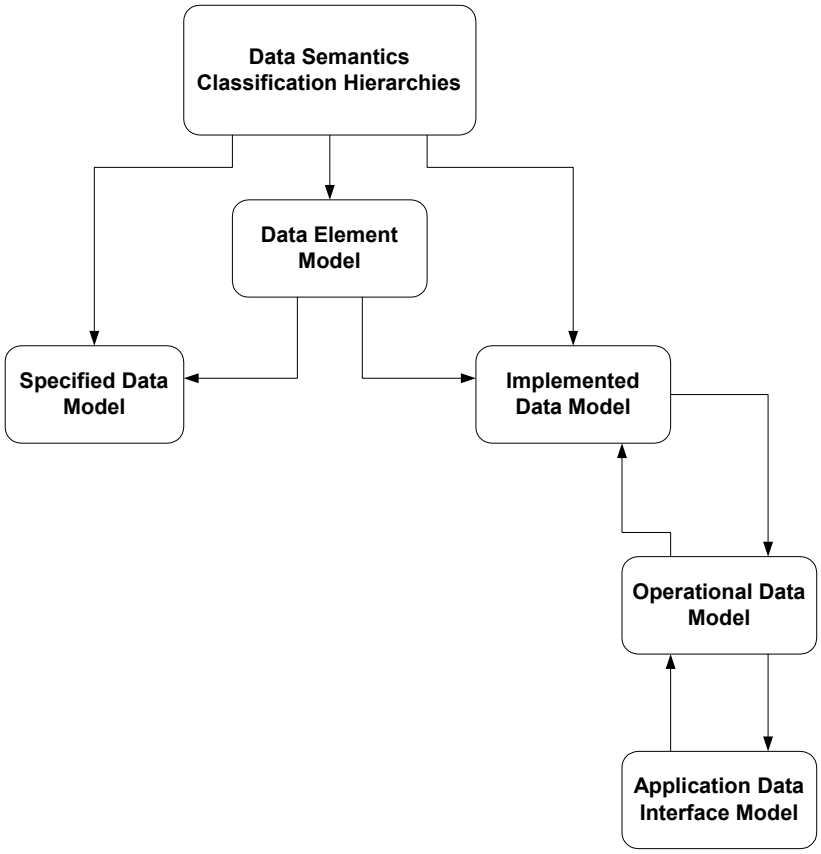

Figure 1. Whitemarsh Data Modeler Six Submodels

attributes within entities, columns in tables, DBMS columns with DBMS tables, and view-columns within Views.

The specified data model is technology independent and thus not bound by any particular database management system or computing implementation, for example, a data structure modeling the needs of human resources (HR).

The implemented data model is technology dependent and is bound by a particular linguistic expression of a data model's semantics, but has not been implemented through a particular DBMS, nor particular computing environment. For example, a HR database expressed in the ANSI SQL:1999 language.

An operational data model is technology dependent, is bound by a particular linguistic expression through a specific DBMS, and is implemented on a specific computing environment to serve the needs of one or more data processing applications. For example, an HR database under Oracle/8 on a Compaq Proliant Unix server in Cleveland, Ohio.

Finally, the application interface model is the application-side data model expressed as SQL views. Thus, the application interface model is technology dependent, is bound by a particular linguistic expression through a specific DBMS, and is implemented on a specific computing environment to serve the needs of one or more data processing applications.

Collectively, these six models enable enterprises to know what means what, where it is implemented, and what applications it serves, no matter under which technology it is implemented or under what name the data is called.

A full presentation of these six models is contained a newly revised version of the Whitemarsh book, *Whitemarsh Metabase: Data Modeler Architecture and Concept of Operations*, which was published in the Spring of 1999. The book is currently a free download from the Whitemarsh website. This *Olde Saw...* article presents a brief overview of the first three Whitemarsh data modeler submodels.

In all Whitemarsh figures, lines with an arrow head represents traditional one-to-many relationships. In Figure 1 then, data semantics classification hierarchies govern one or more specified data models, data element models and implemented data models. The data element model provides the semantics for attributes of the specified data model, and the columns of the implemented data model. The Implemented data model supports one or more operational data models. An operational data model may also be an implementation of one or more aspects of an implemented model. The application data model also supports one or more operational data models, and one application data interface model may be implementation of more aspects of an operational model.

In an ideal world, enterprises would provide "human" data modelers all the time they need. They would start with blank sheets and once the perfect top-down data models were complete, then, through magic, all existing information system implementations of the existing data models would be instantly re-configured. Because such magic doesn't exist, the Whitemarsh data modeler, in addition to its top-down data modeling facilities, embraces bottom-up importation of existing SQL DDL and the inductive building of enterprise data models, one application and one database at a time.

Within the Whitemarsh data modeler, the terms data element domain, data element, attribute, column, DBMS column, and view column are not only spelled differently, they absolutely mean different things. A data element domain represents the semantics that govern a broad range of typed values such as date, time, or money. Data element domains may be hierarchical so that money could be further broken into USD, CDN, British Pounds, etc.

A data element is the semantic template for attribute and column. An attribute of an entity represents the semantics of a technology independent data fact. Attributes have no business value instances, i.e., "John Jones."

A column of a table represents the semantics of a technology dependent data fact. By its very use, the technology is SQL. Columns have no business value instances because the specific DBMS is not known nor are any of the other necessary physical instantiation components.

DBMS columns represent business value instances. Each DBMS column exists within a table of a specific database on a specific platform that is under the control of a specific instance of a DBMS such as Oracle, Sybase, or DB2.

Finally, view columns are the DBMS column semantics known to the application program. View columns also do not represent value instances. They are, instead, mapped to DBMS columns which the DBMS employs to access tables of rows of data. A view column may additionally be a complex type such as full name or address.

Collectively, Whitemarsh refers to data elements, attributes, columns, DBMS columns, and view columns as data components. Every data component has a common business name that stands alone or that is prefixed and suffixed by words that convey additional semantics. The next three sections provide a brief overview of the Whitemarsh data modeler's:

- Data Semantics Classification Hierarchy Submodel
- Data Elements submodel
- Specified data model submodel

2. Data Semantics Classification Hierarchy Submodel

The data semantics classification hierarchy submodel enables the creation of semantic classifications employed to regularize the rules and meanings of the data represented by the data element, specified, and implemented data models. Figure 2 presents the data semantics classification hierarchy submodel. The key entities in this figure are:

- Meta Category Value Type Classification
- Meta Category Value Type
- Meta Category Value

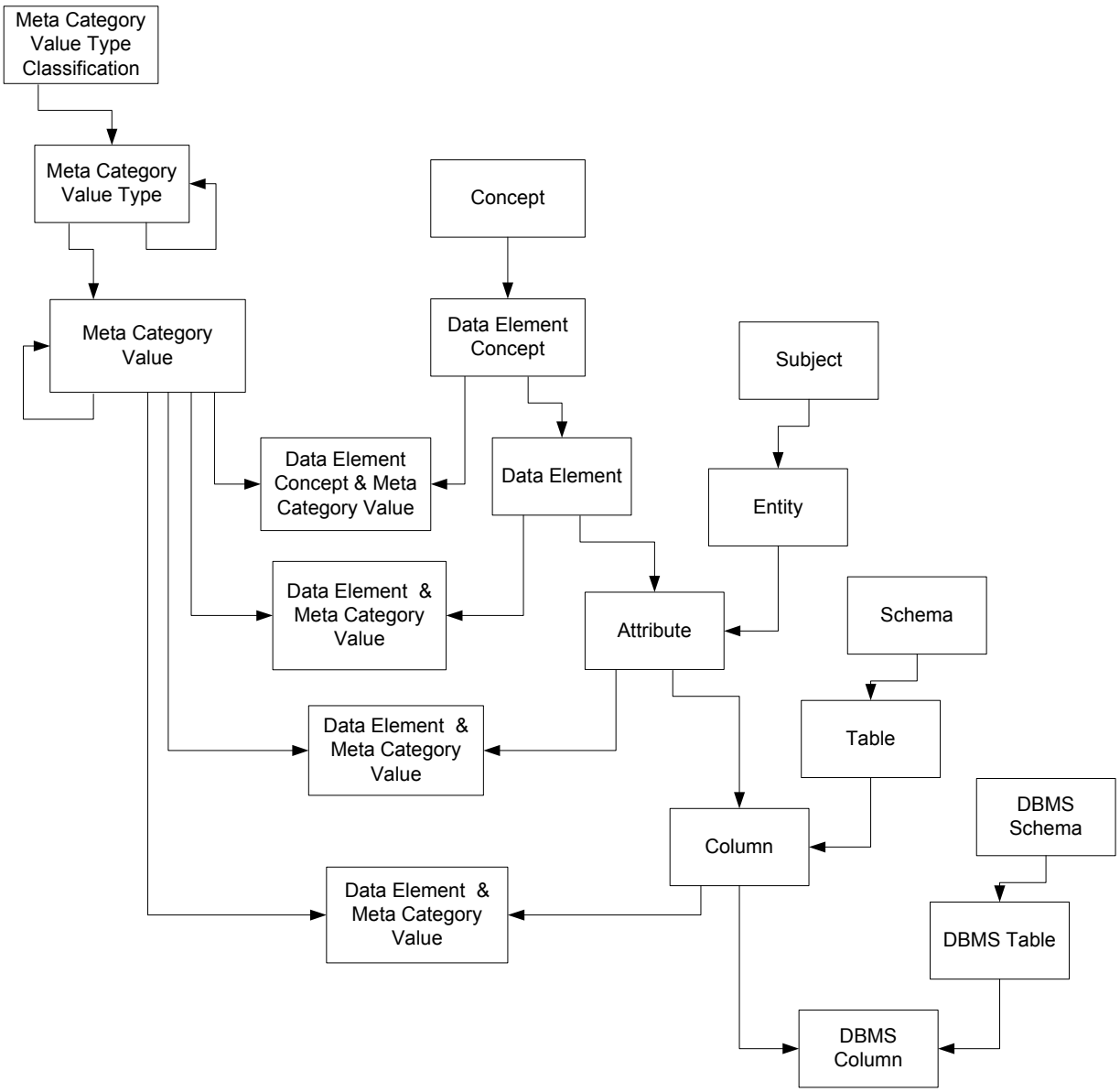

Figure 2. Subset of Meta Entities for the Whitemarsh Metabase's Data Element meta model.

The purpose of Meta Category Value Type Classification (MCVTC) is to separate all semantics that come before the common business name from all those that go after the common business name. There are only two rows allowed in this meta entity. One with the value, prefix and the other with the value, suffix. In general, Whitemarsh refers to all prefix semantics as Semantic Modifiers, and all suffix modifiers as Data Use Modifiers.

The Meta Category Value Type (MCVT) groups the various meta category values according to their type. Figure 3 presents examples of Semantic Modifiers, which are prefix types. In this example, there are four different meta category value types: Temporal, Accuracy, Geography, and Organization. The quantity is up to the enterprise. In this example as well, there is only one level to each the meta category value type hierarchy. This too would be an enterprise standard. The role within the data modeler for MCVT is to segregate prefix types and values from suffix types and values, to specify the order that a type is to appear within the list of semantics within a data component.

The final meta entity, meta category value, contains the actual values for the meta category types. Meta Category values can also exist in a hierarchy. For example, year contains month contains week contains day.

Meta Category Value Type and Meta Category Value Hierarchies				
Meta Category Value Type Hierarchy	**Examples of Semantic Modifiers**			
	Temporal	**Accuracy**	**Geographic**	**Organizational**
Examples of Meta Category Value Hierarchies for the Meta Category Value Types	Last	Estimated	World	World-wide
	First	Projected	Hemisphere	Business unit
	Latest	Revised	North America	Region
	Earliest	Initial	United States	District
	Current	Actual	Mid-Atlantic	Territory
	This year		Maryland	
	Last year		Bowie	

Figure 3. Meta Category Value Types and Meta Category Value Hierarchies For Semantic Modifiers.

The other meta category value type classification, that is, those words that come after the common business word of a data component are Data Use Modifiers. A typical set is presented in Figure 4. Data use modifiers are also represented by same three meta category value meta entities. By tradition, there has only been ONE word allowed after the selected set of modifiers and it was called the class word. That, however, never really made sense as the following examples illustrate. Suppose there is a price table, where PRICE is the primary key column.

Shouldn't PRICE's class word be Identifier? Yes. But, isn't price also an Amount? Yes. Why must we force ourselves to make only one class word choice? How can Social Security Number be an identifier, but not a code, and certainly not it's already existing class word choice, number? But is it really a number with those two embedded hyphens or are they really minus signs? And, in the case of hyphens, the SSN can't be a number. It can only be text, as in SSText. And what about the telephone number, Dial-A-Prayer?

The way out of this "absurd to be in" situation is to recognize that there are multiple classes of words that can come after the common business name, just like there are multiple classes of semantic modifiers. These words, like the semantic modifiers, are grouped into categories and from each category only one choice is made.

When the set of semantic modifiers are prefixed to the data component's common business name and then the data use modifiers are suffixed to form the complete set of semantics for a data component, the following problem can be easily solved. Suppose, as shown in Figure

Meta Category Value Type and Meta Category Value Hierarchies			
Meta Category Value Type Hierarchy	**Data User Modifiers**		
	Data Type	**Role**	**Unit**
Examples of Meta Category Value Hierarchies for the Meta Category Value Types	Date or date component	Identifier component	Day
	Code	Factor	Case
	Text	Flag	Aisle
	Weight	Indicator	Pallet
	Dimension	Identifier component	Transaction
	Money	Rank	Percent
	Integer	Business fact	Inches

Figure 4. Meta Category Value Type and Meta Category Values

5, there were three different columns having the following names:

- Final Sales Amount Northeast
- Northeast Final Sales Amount
- Final Northeast Sales Amount

Meta Category Value Type Classification	**Meta Category Type**	**Meta Category Value**
Prefix	Accuracy	Final
Prefix	Geography	Northeast
Suffix	Data type, qualitative form	Amount

Figure 5. Meta Category Value Hierarchy supporting Final Northeast Sales Amount

The Whitemarsh data modeler's metadata, as presented in Figure 2, "can see" all the column names as the same because each is the same collected set of semantic name and value pairs associated with the common business name, Sales. In the metabase, all these forms are be automatically set to Final Northeast Sales Amount. When reverse engineering from a legacy schema into the Operational Data Model (as illustrated in Figure 2), a DBMS column's existing name string would have to be examined to determine its semantic components. Once identified the component collection can be associated with an existing column. Currently, while the process is manual, it can be tedious, but the value is immense. Once, however, a collection of DBMS schemas is reverse engineered, its common implemented data model can be employed to map to a myriad of view column names that cross all the differently named DBMS columns.

In addition to regularizing the actual prefix and suffix semantic words, missing words can also be determined. For example, what is the type of currency associated with Final Northeast Sales Amount? Assuming there is only one "Northeast" in the world would be presumptuous. But tracing Northeast through its hierarchy to the United States would probably lead to the notion that the amount is in US dollars. It would however be better to specify. Queries can be run against the columns to determine the set of all "amount" columns associated with data elements of the money data element domain that do not have an appropriate currency data use modifier.

If such searches seem remote and possibly uninteresting, just recall the$125 million crash versus fly-by of a satellite into Mars. NASA has stated categorically that the crash was due to a mixup in data semantics (metric vs english units).

In addition to fixed word sets and word order appearance, meta category classification hierarchies can contain these abbreviations (medium, small, and super-small), and definition fragments.

Figure 2 shows that meta category values are associated with data element domains, data elements, attributes, and columns. In such associations two conditions can occur:

- Allocation of multiple semantics from the same hierarchy, that is, multiple allocations of geography to the same data component.

- Allocation of a "same or higher" semantic that is not appropriate, that is, the allocation of Connecticut to an attribute that is mapped to a data element that is already associated with Massachusetts, or the allocation of Connecticut to both.

The first case is automatically prevented through the judicious use of unique key constraints within the metabase. The second is prevented by a software embed that "chases" the hierarchy and when that situation is found, the allocation of the "offending" semantics is automatically prevented.

The definitions and meta category values are especially valuable in avoiding the task of creating business fact definitions altogether, which should be no more nor any less than the stylistic concatenation of the definitions fragments of its semantic parts. The Whitemarsh data modeler provides that facility. The definition fragments are brought together in the same sequence as are the name parts.

A standard report from the Whitemarsh metabase data modeler prints these meta category value types and meta category value hierarchies so they can be reviewed and analyzed against the actual data semantics desired by the enterprise.

Figure 6 shows the screen through which meta category value types are added. The process is simple. Highlight the "parent," such as Data Use Modifiers and then press the Insert button. An update screen appears. Enter the appropriate data and press the Enter key. A new child below the highlighted Data User Modifier then appears. The other buttons on the screen cause the explosion/collapse of a node, or all the nodes. The Print button causes the hierarchy to be printed out. When the screen is employed for selecting purposes the Select button is active. Deletion of a node is possible only if it is not being employed as a semantic any where. Cascade delete is completely disabled.

Figure 7 is similar to Figure 6. In Figure 7, the meta category values are presented. Each line of information consists of three entries: prefix/suffix, meta category value type, and then the meta category value. In the explosion of the Quantitative Form, there are eight shown subnodes. The nodes Amount, Coordinate, Dimension, and Quantity are additionally subdivided. Adding data to this meta category value hierarchy is accomplished the same as with meta category value types. The final set of columns in this figure show the class and types.

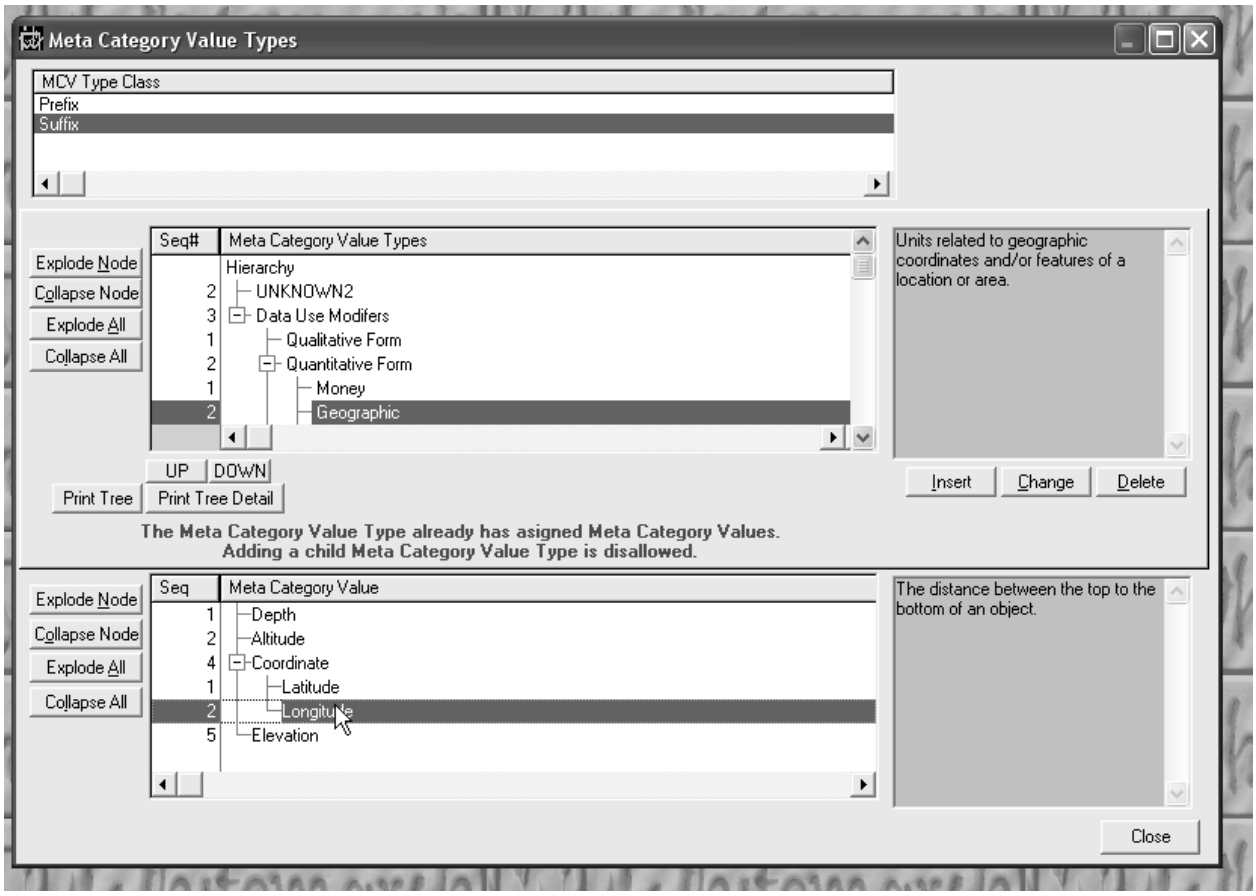

Figure 6. Meta Category Value Types and their associated Meta Category Values.

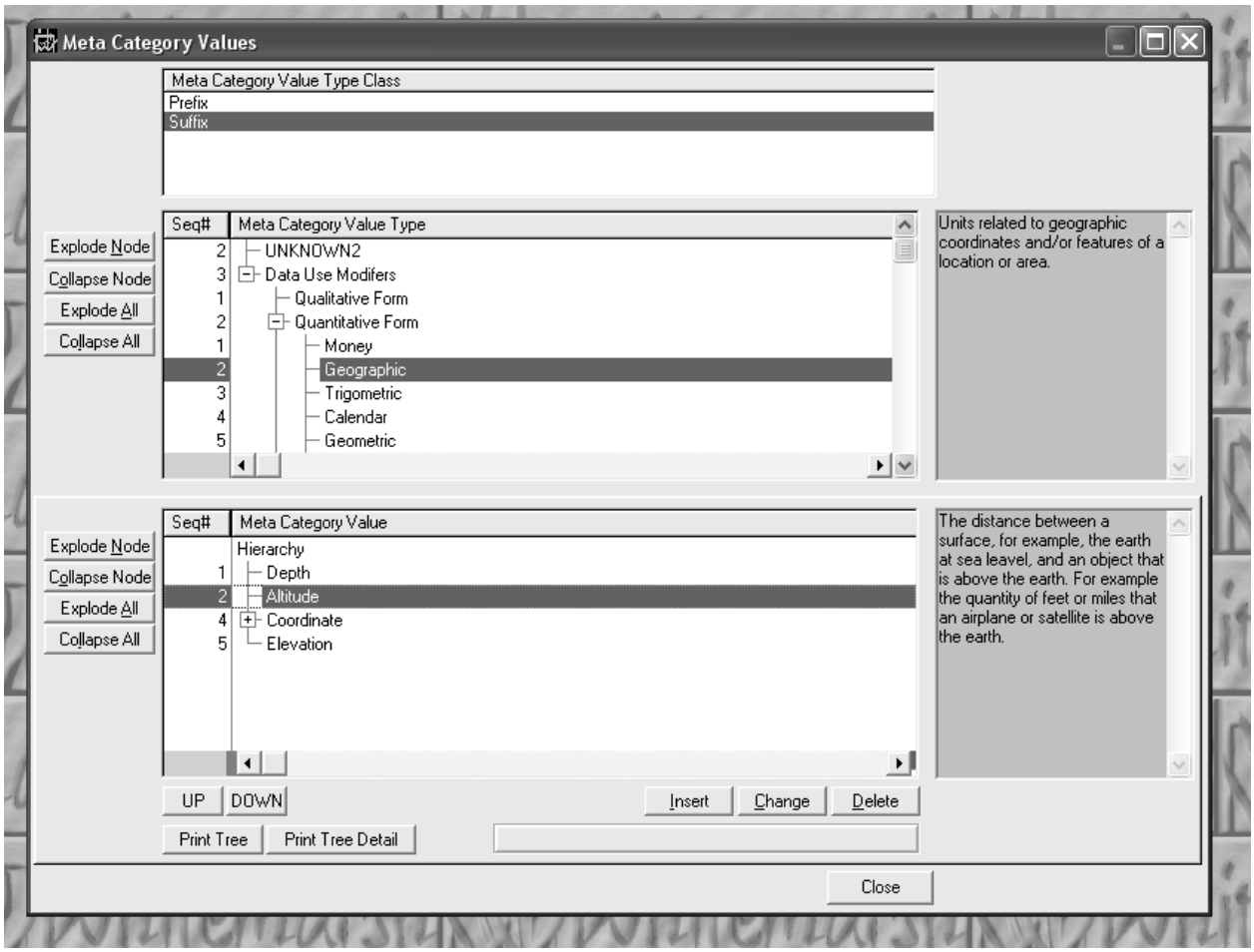

Figure 7. Meta Category Values for a given Meta Category Value Type.

3. Data Elements Submodel

The data elements submodel contains the metadata necessary to support the data element's use as a semantic template in the creation of context dependent collections of business facts. That is, directly with attributes within entities, and columns in tables, and then indirectly with DBMS columns with DBMS tables, and view-columns within Views. In the Whitemarsh data modeler, data elements do not represent business fact values. Rather they represent the semantics of the business facts, which, in turn, represent correctly formed data values.

Figure 8 presents the data element meta model. The meta entities of the model that represent meta category values have been presented above. In addition to data elements, the model includes:

- Concepts, Conceptual Value Domains, Data Element Concepts, Value Domains, and Data Element Classifications
- Compound data elements
- Derived data elements
- Business Domains

Meta category values are assigned to data element domains, data elements, attributes and columns in the same way, tagging. Figure 9 illustrates the process which consists of three parts. An entry is tagged in the upper left window. One or more items are then tagged in the upper right window. Because, in this example, the upper right window is a hierarchy, there are the hierarchy specific buttons that explode or collapse. Once tagged, the Build button is pressed. The result is the set of semantics assigned to the data element. If a semantic attachment is attempted that does not conform to the attachment rules, then the specific data element semantic attachment is rejected. Semantic attachments can be deleted through the Delete button.

Data element domains represent the hierarchical classification of the business semantics of values associated with a data element. For example, if there is a data element, salary, it clearly represents some form of compensation that is received by an employee within the context of the enterprise. While it is true that compensation is a type of money or currency, these are data use modifiers that identify the form of the compensation. For example, the form of compensation could be real property, days off with pay, free travel, and the like. From the point of view of a data element domain, there is first Compensation, its hierarchy of value domains, and then the data element Salary or days-off, etc.

Compound data elements are data elements that are perceived as a unit but have subordinate contained units that are generally "invisible" to the uninitiated. For example, while it's well recognized that a telephone number in the United States consists of a country code, area code, exchange, and then a number, people rarely refer to those specific parts. Rather they say, that their phone number is "1.301.249.1142" as if it were all one continuous string.

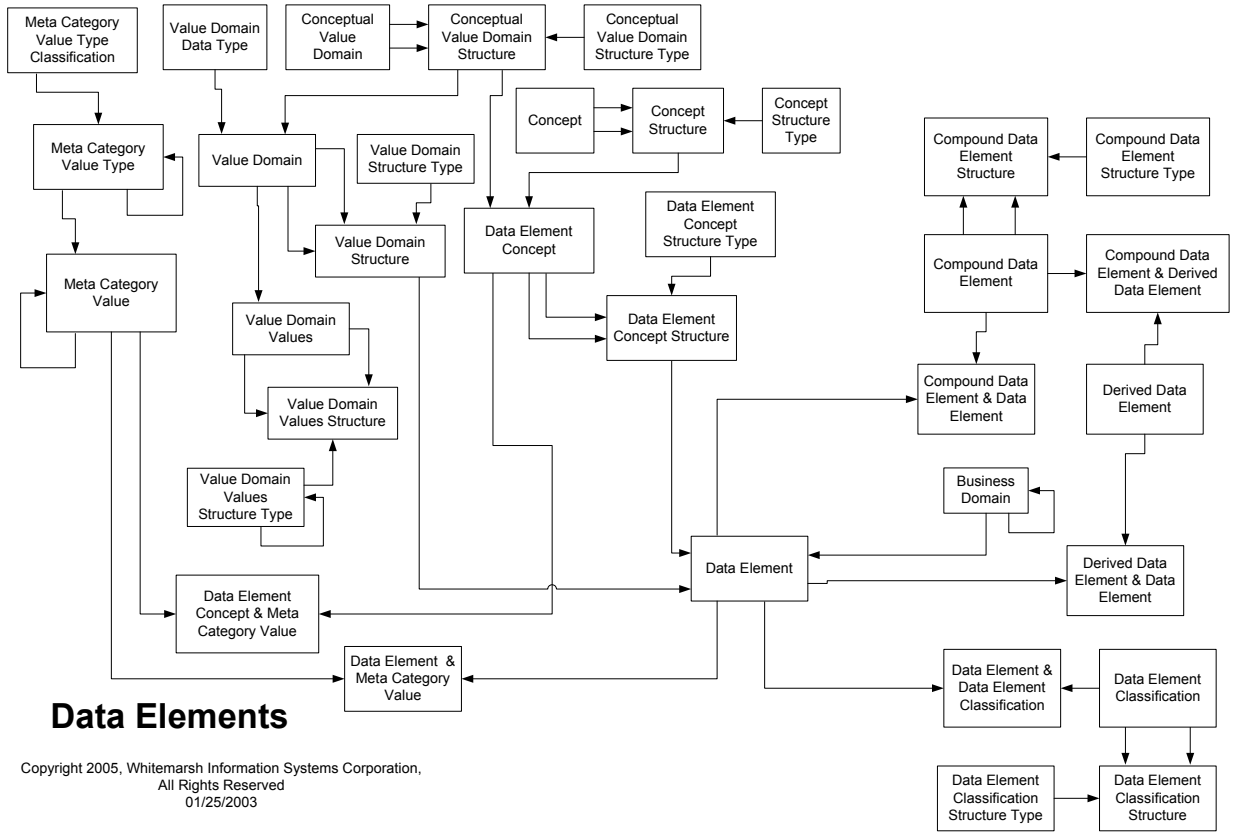

Figure 8. Data Element meta model.

A derived data element is a data element whose value has to be calculated from other data elements. An example could be the *age-at-death* of a person. This value is clearly computed from the difference in terms of years between the *date of birth* and the *date of death*.

Business domains serve to disambiguate the use of the same named data element for different purposes. For example, the data element, Tract, from the dictionary, can mean 1) and area of land, 2) a space of time, 3) a system of body parts, or 4) a bundle of nerves. Another dictionary example, Trade, can mean 1) a course of conduct, 2) a path traversed, 3) a business one practices, and 4) to give in exchange for another commodity. Without business domains, there is no way to differentiate the varied meanings of the same word. Either there can be no data elements with different names across the enterprise, or data elements must exist within business domains, as a way to separate out these semantic homonyms.

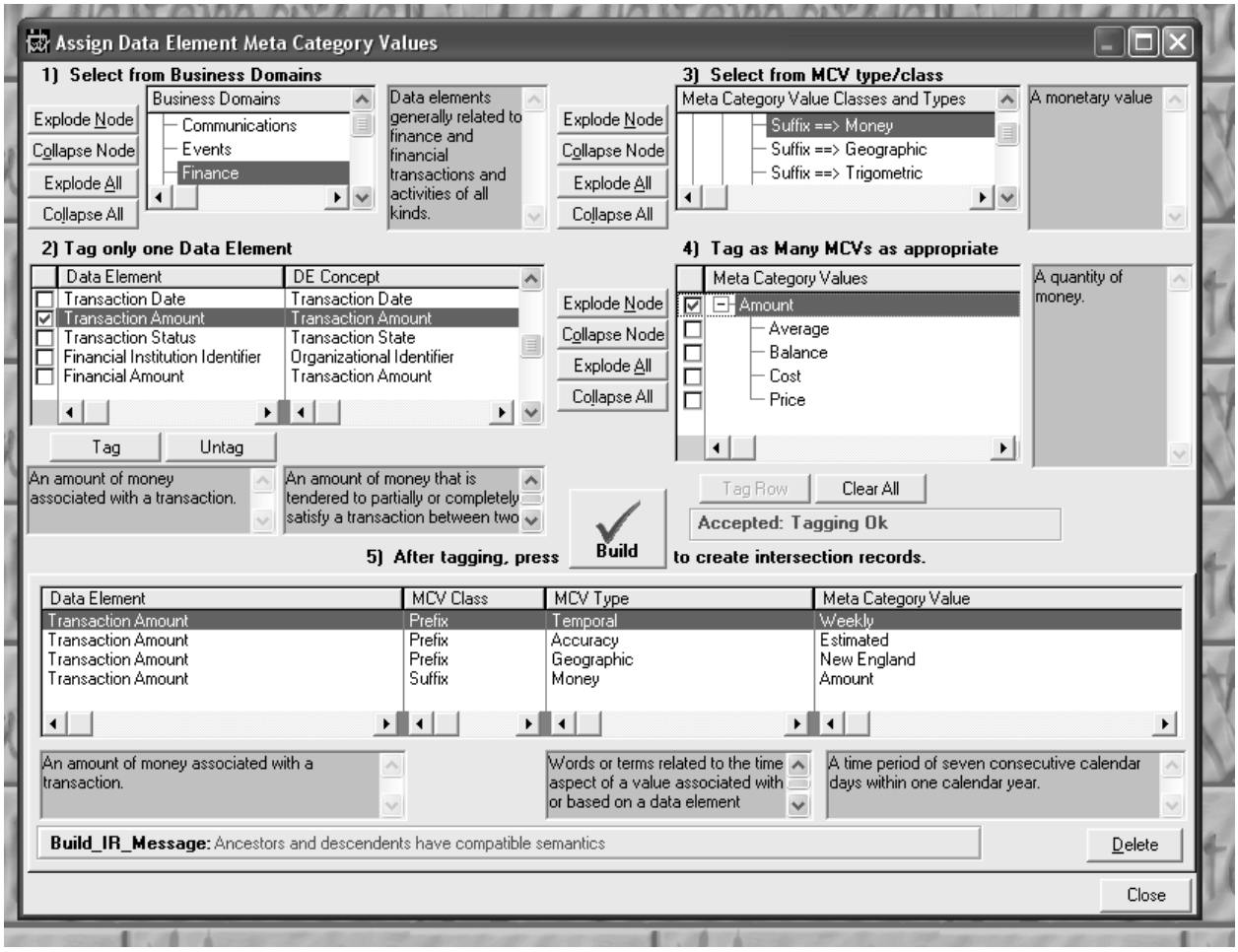

Figure 9. Assignment of Meta Category Values to a Data Element.

Data element value domains permit the inclusion of representative values that can be used to create constraint clauses. In the data modeler, the value domain hierarchy is:

- Data element concept value domain
- Data element value domain
- Attribute value domain
- Column value domain

The value of this approach is compelling. Properly defined, data elements and data element domains can be of great value to the enterprise's mission of achieving data standardization. The following example proves this assertion. If estimates are properly done, then information

technology project managers should allow about one staff hour to fully define each context dependent business fact, that is, a column in a table, a field in a file, a cell in a screen, or a variable in a program.

If there are 100 tables of 15 columns per table, and 200 screens with 15 screen cells on each, and 300 programs with about 30 variables per program, the quantity of these context dependent business facts is 13,500. Thus, the quantity of staff hours that should be expended on context dependent business fact definition should be about 13,500 staff hours. Given that no project manager will ever presume that such an estimate will be accepted, either an unrealistic estimate will be made, such as 15 minutes per context dependent business fact, which brings the estimate down to about 4,000 staff hours, or the context dependent business facts just won't be defined. Since the 4,000 staff hours would also never be accepted, then, in virtually 100% of the cases the context dependent business facts will not be defined.

Clearly, that is not a proper solution to a very real problem. Not documenting the various business facts in all their contextual uses is professionally irresponsible. Y2K, and the Mars failed "fly-by" are perfect examples of the consequences of ignoring data standardization.

Given that there are only about 20% "real" data elements across a population of columns in tables, then the 1500 columns is immediately reduced to just 300 data elements. If two hours is expended for each, then that's just 15 staff weeks. But given that a database is largely a semantic clone of a number of other databases, then the population of 300 data elements probably already has 75% defined. That reduces the number to just 75 data elements, or about 4 staff weeks.

Since virtually all the other context dependent business facts are uses of already defined columns in tables, which are now almost automatically defined through the 4 staff weeks for defining the few data elements not yet defined, then the data definition problem is almost largely non-existent. Now, that's a proper solution to the very real and serious problem.

Finally, through this approach, since the actual work is so completely integrated into the natural activities of analysis and design, the actual time to accomplish all this data standardization effort will actually be NEGATIVE because of the normal acceleration in productivity due to reductions in:

- Research
- Rework
- Presentations
- Documentation preparation

Simply stated, there are no down-sides to this approach. It decreases risk, decreases cost, increases productivity, and increases quality. The approach is all benefit at virtually no cost. In short, it is work accomplished at the rate management expected it to be done in the first place.

4. Specified Data Submodel

The specified data model illustrates the use of data elements in constructing data models. The Specified Data Model represents the technology independent representation of a set of data structures considered important to the enterprise. In addition to all the appropriate requirements for quality data models, the key requirement for the specified data model is that it can be implemented through a variety of technologies such as DBMS, spread sheet tables, traditional access file structures, computer program embedded temporary files and memory arrays, and the like.

The meta model design for the specified data model is presented in Figure 10. This diagram contains the following meta entity groups:

- Meta Category Value Types and Meta Category Values (upper left)
- Data Element Domains and Value Domains, Data Elements and Value Domains (middle and right)
- Subject, Entity, and Attribute (middle left)
- Primary and Foreign Key Support for Entities (lower left)
- Attribute Value Domains (lower right)

Shaded meta entities are not allowed to be updated within the specified data model module. Rather they are accomplished in the data element module. The reason these functions are segregated is to prevent ad hoc data modeling behavior by allowing separate security over each distinct module. In case there is a need to map to a data element that's not already defined, the Whitemarsh data modeler supports the definition of new data elements, but only through the appropriate data modeler module. Notwithstanding, the attributes can still be added, but when they are without the appropriate data element mapping, reports flag these unmapped attributes.

Subject areas, within the context of the specified data model is a method of classifying entities. For example, the subject area, Finance might contain two subordinate subject areas, Accounts Payable and Asset Management. Subjects can be hierarchical. Entities are tied to the leaves of the subjects. Entities are thus directly tied to Accounts Payables or to Asset Management, but not to Finance.

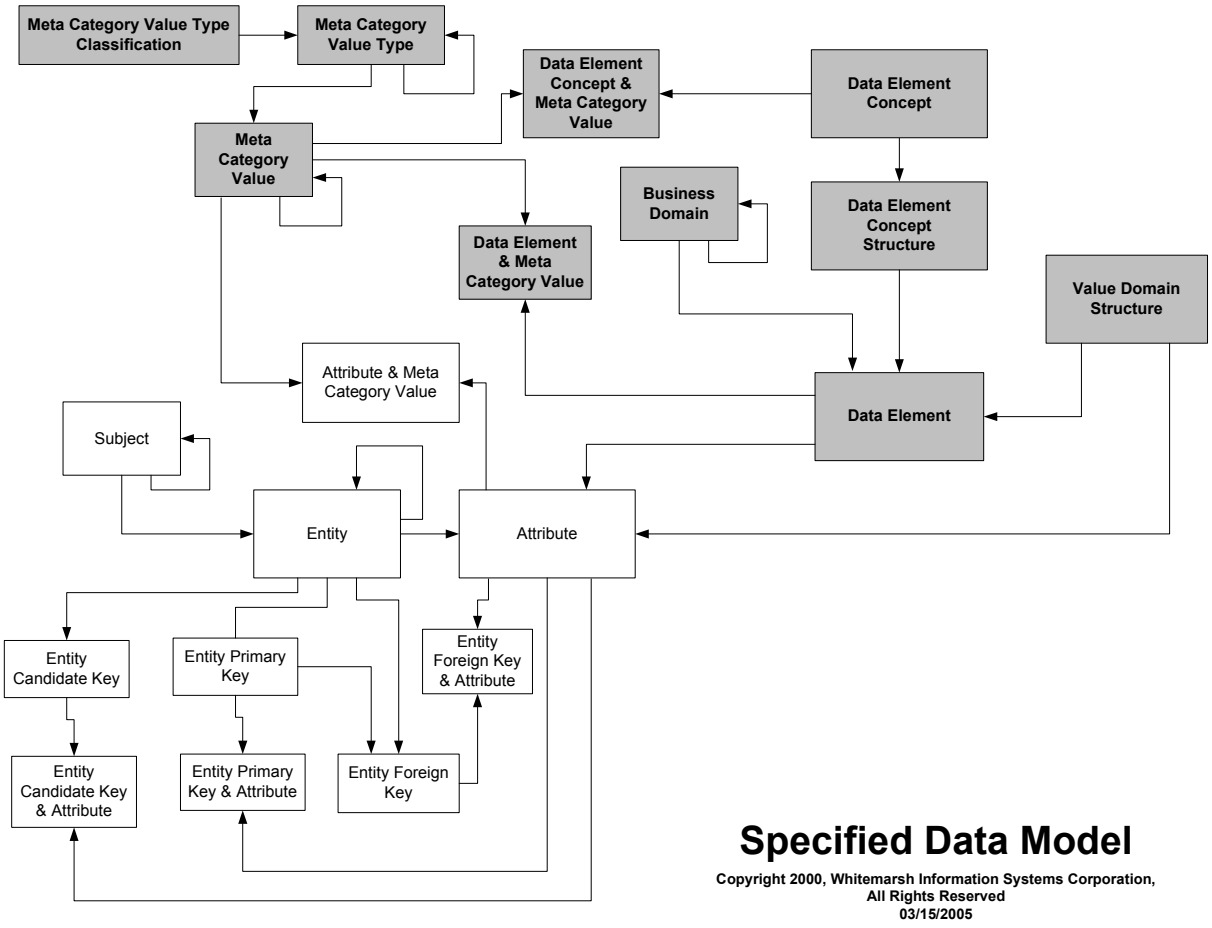

Figure 10. Meta Model for the Specified Data Model.

While most all the Whitemarsh data modeler material to this point concentrates on data elements and the semantics attached thereto, the most valuable component of a data model is the entity. An entity should represent a well defined and obviously recognizable component of business policy that is instantiated, modified, selected, and reported. Entities are of course recognized through the data values represented by its contained attributes, hence the need for business policy rooted and easily recognizable data element semantic template from which to model the attributes.

While entities are identified within a specific subject area, it is merely a mechanism for classifying the entities. Entities from different subject areas can be interrelated through primary and foreign keys. Thus, all entities directly and indirectly associated with a particular subject area can be reported.

Once the subject areas and entities are identified, the task of creating attributes starts. At this point, the basic definition of data element comes into play. That is, a semantic template for the business fact attributes within an entity. Attribute definition is accomplished in two stages:

- Employment of the data element semantic template along with the entity to create an attribute.
- Refinement of the attribute's meta attributes through the creation of its remaining meta attribute values.

This first stage is accomplished through tagging. Figure 11 presents a four list window. In the upper left list is the list of entities. In the middle right window is the list of data elements. Data elements appear within the Business Domain hierarchy context. As a business domain is selected, the appropriate data elements appear. An entity is tagged and then one or more data

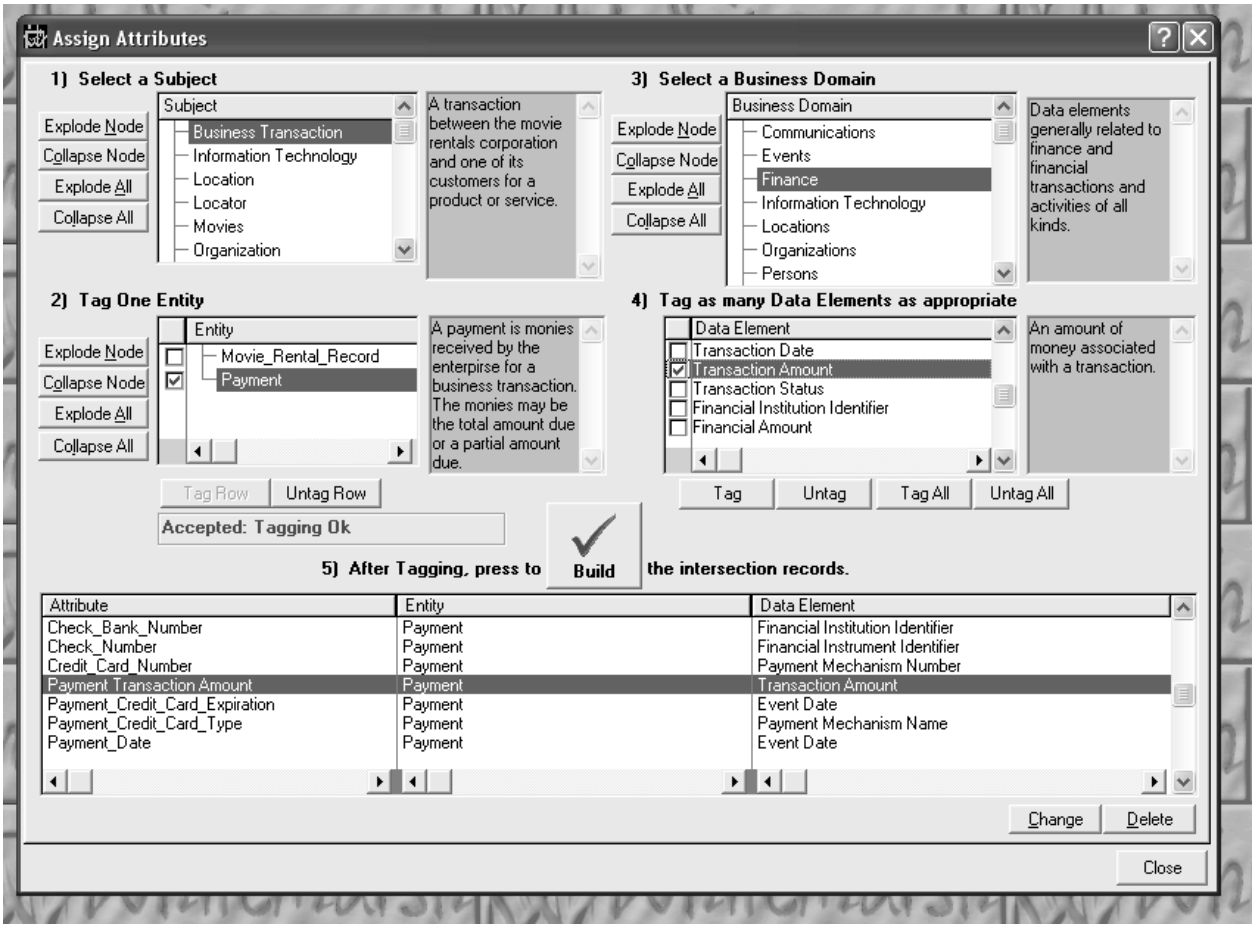

Figure 11. Tagging window for creating an attribute from a data element within an entity.

elements are agged. Once the "build" button is pressed, the attributes are built and displayed in the bottom list that runs across the full screen. That's all there is to it. Because of inheritance, all the semantics of the data element are automatically assumed inherited.

Once an attribute is built, its semantics can be enhanced through the allocation of a user-set name, semantics and a description. Whenever semantics are added to an attribute they are automatically checked to ensure they are not in conflict with those already allocated or inherited through the attribute's data element.

Relationships between entities are created through primary and foreign keys as depicted in Figure 12. The process consists of seven distinct steps. First, identify the source entity's primary key. This is done through selection from a list. Second, identify the target entity. Third, enter an action phrase. Fourth and fifth, select the appropriate key match and referential action choices. Sixth, describe the business policy basis for the relationship. Seventh, press the button. When the button is pressed, the source entity's primary key attributes are employed to create new attributes within the target entity. The foreign key name as well as the foreign key column names are automatically constructed. Foreign key attributes are automatically allocated the full set of semantics allocated to the primary key attributes. Multiple foreign keys can be created between two entities so long as the action phrase is different. The data modeler automatically prevents multiple allocations of the delete referential action, Cascade.

Because of these work saving assists, creating a specified data model is an effort that is characterized by:

- Significantly shorter times to create entities because of all the inherited semantics
- Lowered risk because when things are the same they will be semantically defined the same
- Increased quality because there will be more time for important semantic component parts
- Increased productivity because the process of creating the specified data model can be accomplished by functional experts rather than just data administration staff.

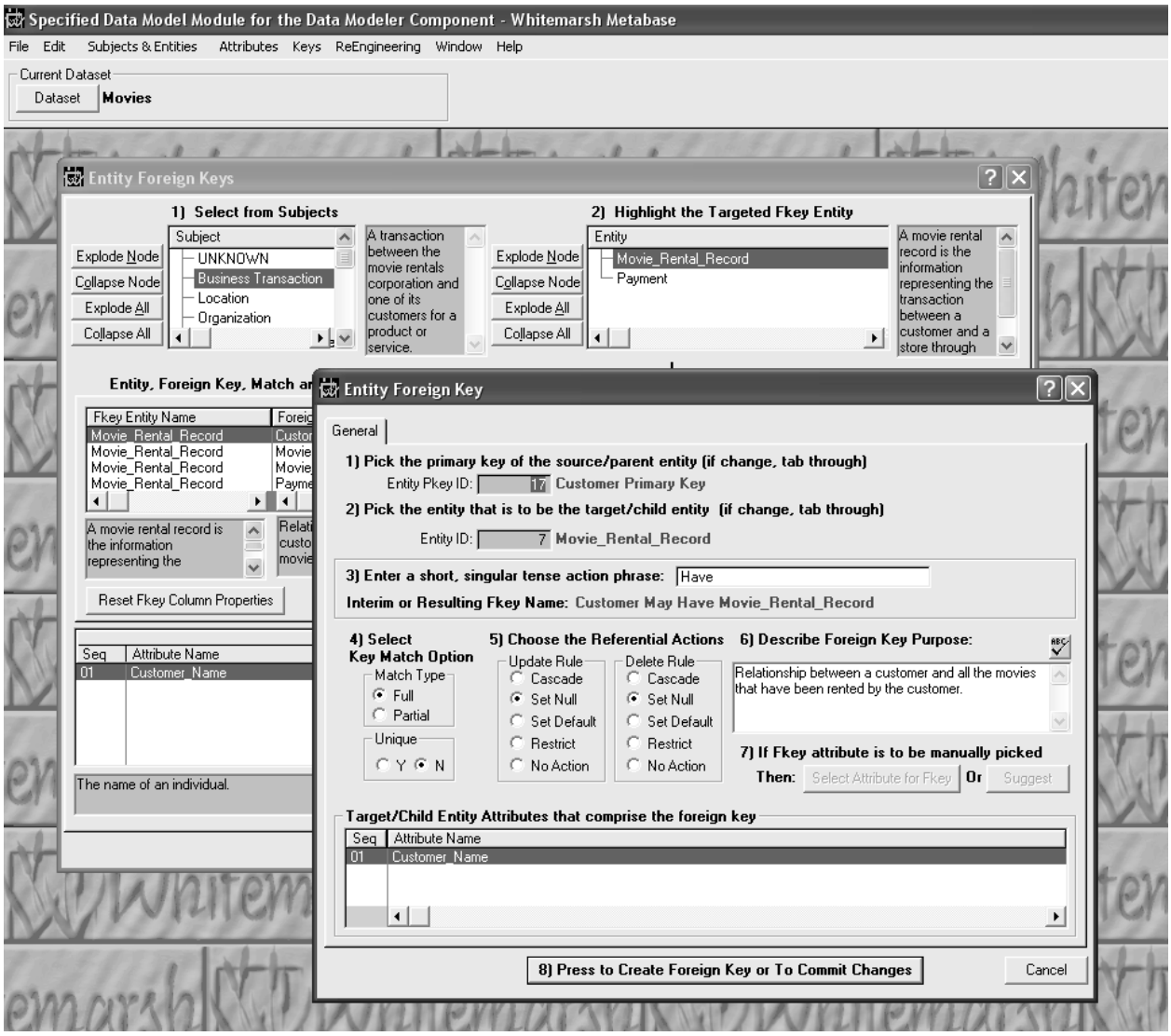

Figure 12. Relationship creation process through foreign keys.

5. Olde Saw Summary

The Whitemarsh metabase component, data modeler goes a long way to solving the problems identified in the March 1998 Olde Saw article which also has been recently updated. To wit:

Tooth One: With the Whitemarsh data modeler, there is now a clear and practical way to make data elements the semantic templates for attributes, and columns, and via columns, DBMS columns. Through DBMS columns, data elements also become the semantic templates for screen cells and report fields. Across the enterprise data elements are generally less than 5% of the total count of attributes, columns, DBMS columns, view columns, report cells, and screen columns.

Tooth Two: The Whitemarsh data modeler now enables data elements to have names. Data elements can not only have their own names, their names, when the same as other data elements are able to be disambiguated through their assigned data element domain, business domain, and the assigned meta category values. When all this is done, the only thing that will be the same is the name.

Tooth Three: The Whitemarsh data modeler has kept data component names as a separate meta entity attribute. Thus, while the name is automatically materialized, there can be two other names as well: the common business name and the user-set name. Notwithstanding the three names, all the semantics that comprise the data component are kept separate. This means that even though the user-set name may have been changed, the overarching semantics never change.

Additionally, the name can be shorted through the use of medium, small and super small abbreviations. Given that most software products have virtually removed any name length restriction, there should be no problem in the name at all. Finally, data modeler permits the inclusion of definition fragments for each semantic value. Thus, when the full description of the data component is produced, its definition consists of all its assigned meta category value semantics and also all its inherited semantics.

Simply put, the Whitemarsh data modeler may have just eliminated the need for any attribute, column, DBMS column, or view column definitions more detailed than the common business name coupled with the definition fragments automatically supplied by the assigned and/or inherited semantics.

Tooth Four: Semantic modifiers are now completely segregated into different types such as accuracy, geography, temporal and the like. Enterprises can define their own modifier hierarchies and for each value provide its meaning. Then, whenever one is used, its meaning is also provided. The Whitemarsh data modeler enables enterprises to define the order of modifier use. Because the Whitemarsh data modeler is fundamentally a database, queries can be launched to know which data components employ certain semantic values and which are missing.

An interesting use of the semantic modifiers is to turn them from name parts into values. For example, instead of having DBMS column called Northeast Estimated Sales Amount. The

table could have a column called sales amount, and then peer columns for Geography, and Accuracy. A given row would then have the column values, Northeast, Estimated, and $12,500,000. What then is one database's types becomes another database's instances. This strategy is squarely based on the meta category value hierarchy approach. A natural spin-off from this use is the creation of XML based data for shipping between sites. Since XML is simply text based hierarchies of types and instances contained within a recognized start and end markers, the approach described here fits the needs of XML perfectly.

Tooth Five: Finally, the data modeler does away with the unjustified notion that there can only be one class word. Thus, a Social Security Number can have the data use modifiers of identifier and number in one situation, and business fact and number in another situation. Similarly, if part of a primary key is transaction date, it can be both of the date data type and its role can be an identifier.

In conclusion, the five "teeth" of the olde saw have certainly been addressed. I now leave you again left with the same admonition, *That old saw just don't cut it. Never did, and never will. So, stop wasting your time and money and choose one that does.* Whitemarsh believes that it's data modeler is an acceptable response to the challenge put to us, *Put our software were our pen was.*