Easing the Transition between Attribute-Value Databases and Conventional Databases for Scientific Data

Prakash Nadkarni, MD and Luis Marenco, MD
Yale Center for Medical Informatics, New Haven, CT

Abstract: *We have previously developed and described a modeling and development framework called EAV/CR, which is appropriate for designing databases containing highly heterogeneous and evolving data, as in the case of scientific databases for rapidly advancing domains. The use of EAV/CR has been hampered by the lack of generic tools for non-procedurally transferring data into or out of legacy systems or analytical packages: the transfer task is complicated by the different representation of EAV vs. conventional data, which is not addressed by commercial data-transfer programs. We have therefore created such a tool, which works with a wide variety of data sources that are accessible via Microsoft OLE DB technology. The data transfer tool requires minimal programmer intervention to set up, and no programming to use on a regular basis. Current limitations of the tool are also noted.*

INTRODUCTION

An Entity-Attribute-Value (EAV) design is useful for managing highly heterogeneous data within a database. EAV is used extensively for clinical data, where the number of potential parameters that can apply to a patient across all specialties of medicine is typically much greater than the number of parameters applying to an individual patient [1-3]. In EAV design, we conceptually have a table (or datatype-specific tables) with three sets of columns: an Entity or Object (e.g., the patient being described, along with one or more timestamps), an Attribute (the parameter being recorded), and a Value (of the parameter). As opposed to a complex traditional schema where facts may be segregated into dozens or even hundreds of tables (e.g., organized by clinical specialty), EAV designs readily support generic browsing interfaces where the commonest type of query is "tell me everything about this object (patient)." For rapidly evolving scientific domains, an advantage of EAV design is schema stability as knowledge evolves: only the metadata (the developer-supplied data describing the conceptual database schema) needs to change.

Clinically related EAV implementations have generally assumed that the values are atomic or *simple*. For scientific data, however, one must support values that are IDs of objects (which themselves have attributes and values, nested to an arbitrary degree of complexity.) Our model to support *complex* values (classes), which overlays object-oriented concepts on to the basic EAV model, is called EAV with Classes and Relationships (EAV/CR) [4]. This model was originally created for SenseLab [5], a neuroscience database that is supported by the NIMH's Human Brain Project and Web-accessible via http://senselab.med.yale.edu.

Despite its name, EAV/CR permits conventional tables to coexist with EAV tables. (Certain classes of data, e.g., patient demographics and genotyping, are inappropriate for EAV representation because the same facts, or types of facts, are recorded for all patients.) However, the metadata describing classes and attributes is extremely detailed, going well beyond the built-in metadata ("data dictionary") of database management systems (DBMSs). The metadata is the basis of generic routines that automatically generate robust Web-based interfaces, as previously described in [6]. It must therefore record schema-type descriptions as well as how classes are physically represented, how individual attributes are displayed and how the user interacts with them. The generated interfaces adhere to the principle that end-users should not know nor care what EAV is: to them, all data should appear as if it were conventional.

A well-known drawback of EAV design is that much of the functionality of DBMS-vendor or third-party development tools must be reinvented, because these tools were never designed for EAV data. One such function is data conversion. When a large volume of conventionally structured data already exists, conversion to EAV/CR, which requires creating the metadata followed by data conversion, can be laborious. Further, data conversion is not a one-time process: e.g., SenseLab periodically receives bulk submissions from collaborators with existing electronic data, and one must not force data reentry on them. Periodic data export is also needed: it is well known that EAV data must be transformed to conventional (one-column-per-attribute) format before it can be handled by statistics or graphing packages. Conversion of EAV data to conventional structure involves transforming rows into columns, in a manner similar to generating the "cross-tabs" of spreadsheet packages. The reverse process is somewhat simpler in concept: complications arise,

1067—5027/01/$5.00

however, due to foreign-key references, as discussed later.

In a research consortium where programming expertise may not be widely distributed (and where many labs use spreadsheets to manage data rather than DBMSs), the onus of bulk data conversion lies with the team that is administering the shared database. Clearly, writing very similar per-request conversion programs is not an efficient use of this team's resources. In this paper, we describe a generic tool for bi-directional data conversion, which is scheduled for imminent production deployment in SenseLab. (The tool will be distributed freely to anyone making a written request.)

METADATA CREATION: SCHEMA EXTRACTION

To function as desired, an EAV/CR application is critically dependent on the correctness of its metadata (in terms of the designer's intentions). When populating an EAV/CR database for the first time from an existing conventional database, the first step is to create the schema metadata. The tables, columns and rows of a conventional database are equivalent to the classes, attributes and objects of EAV/CR, and it is highly desirable to capture basic class/table definitions as well as inter-table relationships automatically. This process, *schema extraction*, reduces the developer's workload significantly. (Note that some metadata, e.g., specification of how attributes are to be displayed, must be manually specified in a second pass.) Because every DBMS vendor's data dictionary has a different structure, we use Microsoft's Active Data Objects Extensions for Data Definition and Security (ADOX) to achieve a measure of portability in our schema extraction code. ADOX currently has some limitations, e.g., one cannot determine whether a primary key field in a table is auto-incremented each time a new record is inserted.

A practical point here is that, when someone else has developed the conventional database, it is strongly recommended that one critically inspect and understand its schema. EAV/CR does not exempt the developer from applying the well-known principles of database normalization: it is only the physical storage mechanism that is different from a conventional database. Any flaws in the conventional design would therefore translate into a flawed EAV/CR design.

TRANSFERRING THE DATA: MAPPINGS

Once metadata has been defined, data transfer can operate in one of two modes as previously described: one-time (import) or repeated (import-export). In either case, it is highly preferable that this operation should require minimal or no programming expertise. Our tool relies on a special kind of metadata called *mapping metadata*, which stores the correspondence between classes/attributes in the EAV/CR schema and tables/columns in the source/destination data. (For repeated data transfer, this metadata is created by data administrators or lead users through the interface illustrated in figure 1. For one-time import, it is automatically generated by the system.) Note that, in the repeated-transfer scenario, the names of the two respective sets may not correspond exactly, and one may not necessarily want to import all columns, or export all attributes. The point-and-click interface therefore allows correspondence to be specified visually. In circumstances where many names do in fact match, the interface provides an option, illustrated in the figure, to automatically map tables/columns to classes/attributes with the same name. We use Microsoft's OLE DB to extract table/column names from the source data: OLE DB allows access to non-relational data sources such as XML, spreadsheets, or even tab-delimited files. (For the last two, column names must be specified in the first row of the data.)

Populating the Object Dictionary

One requirement of EAV/CR databases is that common information on all objects across all classes (such as name, brief description and the class an object belongs to) be recorded in an Objects table, which is shown on the left of figure 2. (The figure is discussed in detail later.) This *Object Dictionary* (OD) approach was pioneered in bioinformatics by Tom Slezak in 1992 for the chromosome 19 project at Lawrence Livermore Labs [7]: it is also used by the NCBI family of databases (Genbank, PubMed, OMIM, etc.) Information specific to the class of the object may be stored either in class-specific tables or as attribute-value pairs if EAV is used.

In the presence of highly heterogeneous data, the OD approach confers a significant advantage when creating a browsing-oriented search service intended for both end-users as well as external Web applications that can potentially hyperlink to this database. It provides a single, centralized point of reference for a user to search for an object by ID, name, description, keywords, or synonyms, irrespective of which class it happens to belong to. It is possible to program a "dispatcher" that operates on this table to implement the browsing operation "tell me everything about this object". The implementation details– the "Display Method", in object-oriented parlance–depends upon the object's class, but the details are transparent to the external

calling application, which only needs to reference the object's unique ID. The object's data may be returned in program-code-friendly "pure-content" formats such as XML or as human-friendly formats like HTML, which combine content with formatting markup.

Before the attributes of an object can be imported into the schema, the object itself must first be created so that its ID can be referenced elsewhere. In EAV/CR, object IDs are referenced in two places: as the "Entity/Object" column of EAV tables (illustrated with the "EAV" prefix at the bottom of figure 2), and as the "Value" column in the EAV_Objects table. The role of the latter is now described. In a conventional schema, individual tables/classes are related to each other through foreign-key / primary-key links. In EAV/CR, foreign key values are represented as values that contain Object IDs. (That is, all foreign-key attributes point to a single place, the Objects table.) When importing data from a relational database in conventional form, it is necessary to transform the original foreign key values of the source data into the newly created object IDs of EAV/CR. Import of data from a multi-table database must therefore occur in two phases. In the first phase, the name and/or description of every object in every table is imported, and the common object information is created. (The buttons "Set as Name" and "Set as Description" on fig. 1 allow specifying of which columns in the source table serve these functions.)

The original primary key value of every row is also stored in the Objects table, in order that we can programmatically perform a lookup of the Objects table whenever we encounter a foreign key value of a particular class in the source data. (Since primary key values can be of any data type, they must be converted into least-common-denominator string form: in case of multi-field primary key values, the value of each field is concatenated with a non-printing character. The mapping metadata, described later, stores the list of fields that constitute the primary key of every table that is to be imported, so that the key value may be programmatically composed for each row.)

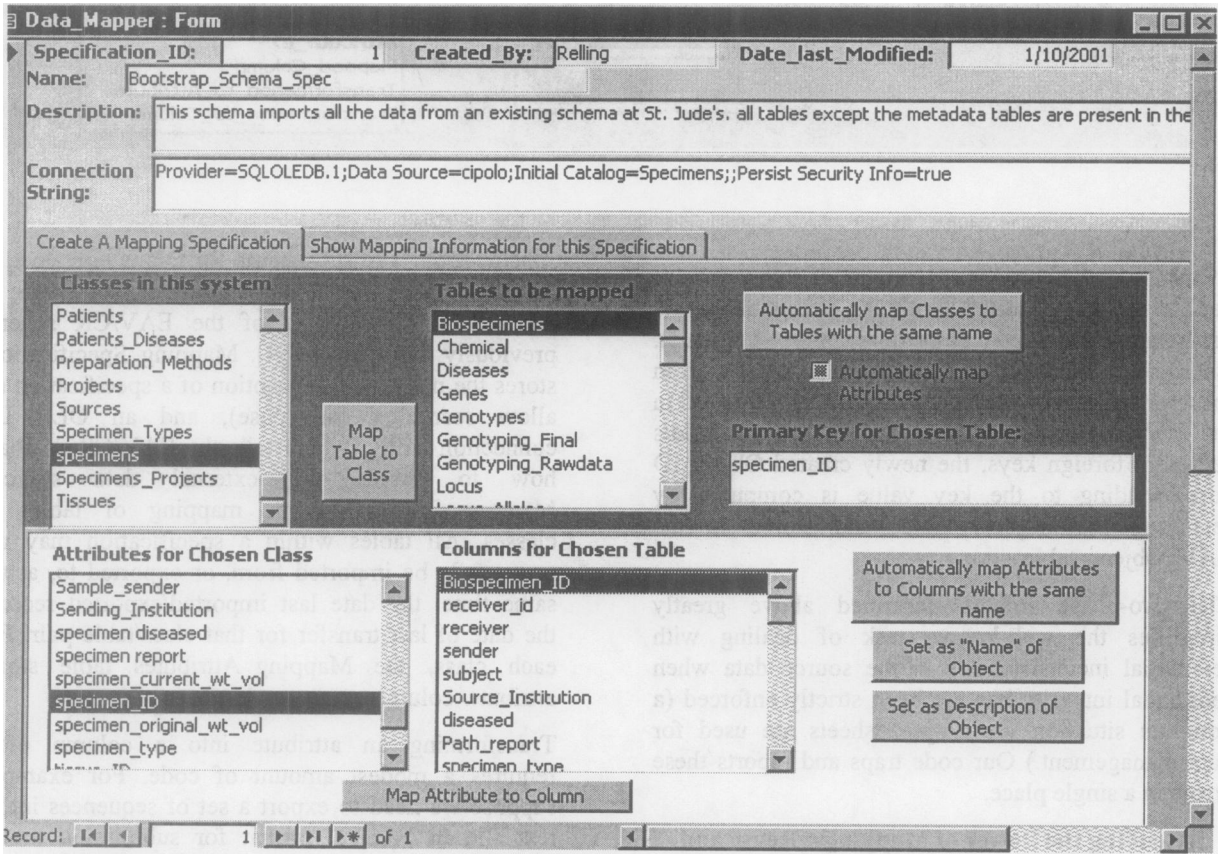
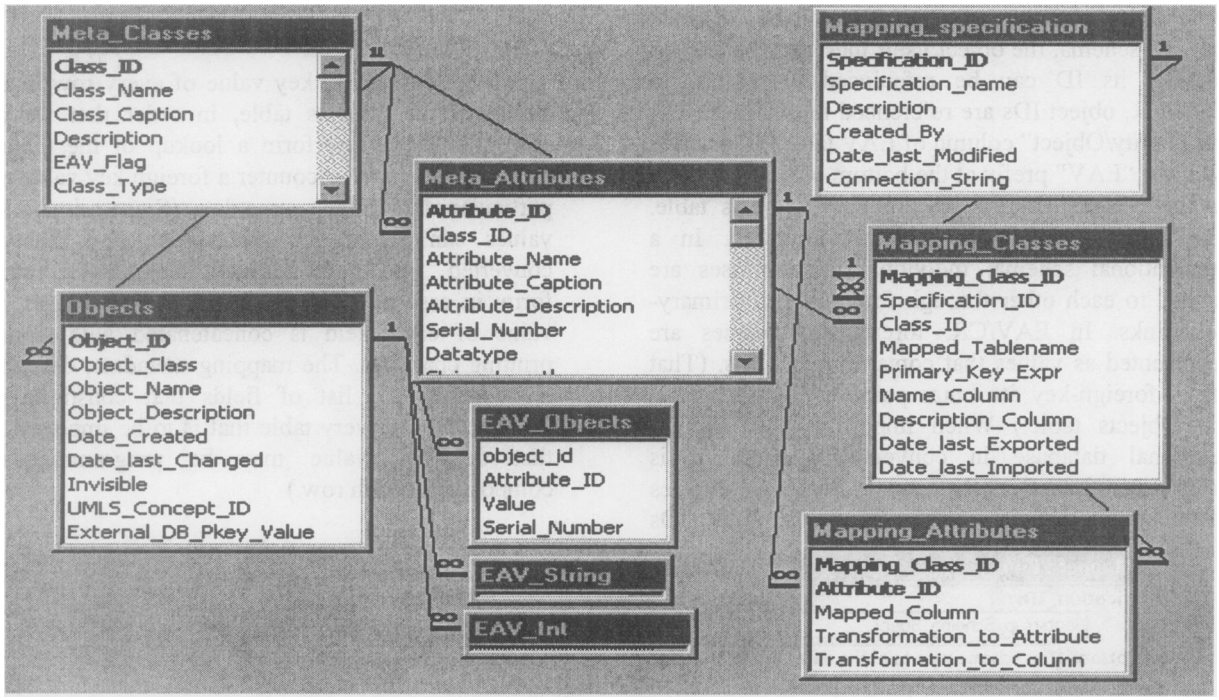


Figure 1: **Creating a Mapping Specification: User Interface**. Within a specification, tables in external data are mapped to classes in the database, and columns in the external table are mapped to attributes in the database. A specification can be saved for later reuse after it is created, and a saved specification can be modified if needed.

Figure 2: **Schema used to support data transfer.** Tables labeled "Meta-" indicate metadata describing the system and are used during data display, entry and form generation. (All details of these tables have not been shown.) The Objects and EAV- tables hold the data: the Objects table holds the name and description of the object, as well as the primary key value of the object in the external data (if it was imported). The EAV tables hold details of the attributes. (All EAV tables have not been shown.) The tables with the "Mapping-" prefix record mapping metadata in the form of specifications, which are stored for later reuse



In the second phase, the attributes of every row in each class are imported (with the exception of the previously imported name/description columns). Based on the datatype of each attribute, it is stored in the appropriate table (e.g., integer values are stored in the EAV_Int table of figure 2). In case the field/s represent foreign keys, the newly created Object ID corresponding to the key value is computed by searching the Objects table, and stored in the EAV_Objects table.

The two-phase import described above greatly simplifies the well-known task of dealing with referential inconsistencies in the source data when referential integrity has not been strictly enforced (a common situation when spreadsheets are used for data management.) Our code traps and reports these errors in a single place.

Extending the Power of Mappings: Reuse and Transformations

Once a set of mappings has been created for several tables from an external source, we permit the saving of all those mappings together as a single *specification*. The subschema for this is contained in the tables with the "Mapping" prefix of figure 2, which also shows part of the EAV/CR schema previously described in [4]. Mapping_Specifications stores the name and description of a specification (to allow searching for reuse), and an OLE DB connection string (which tells the EAV/CR database how to access the external data source). Mapping_Classes records mapping of tables to classes. All tables within a specification may not necessarily be imported from, or exported to, at the same time: the date last imported/exported records the date of last transfer for that class/table pair. For each class, the Mapping_Attributes table stores attribute/ column correspondence.

Transforming an attribute into a column often requires a modest amount of code. For example, suppose we need to export a set of sequences into a text file in FASTA format for submission to the BLAST service of the National Center for Biotechnology Information (NCBI) for determining similarity of the sequences to known sequences in Genbank. This format involves placing a ">" symbol

before the sequence name, a new-line after the name, and a new-line after the sequence itself. Similarly, importing data may also require some data conversion. The columns "Tranformation to Column" and "Transformation to Attribute" in the Mapping Attributes table store Visual Basic expressions that record the nature of the transformation in each case. (Creating these expressions is the only aspect of the tool that requires programmer intervention.) These expressions are evaluated dynamically at runtime using the powerful EVAL() function, which evaluates data as though it were code, executing it and optionally returning a value. The expression can be arbitrarily complex, and can contain calls to built-in and programmer-defined functions.

IMPLEMENTATION: LIMITATIONS, FUTURE DIRECTIONS

The data transfer tool is implemented on the Windows platform with a Microsoft Access client. (Since its use is to be restricted to the database's administrators, there is little to be gained by making it Web-accessible, even though SenseLab itself delivers content over the Web.) The back-end SENSELAB database is implemented with Oracle: it should be readily portable to other DBMSs, since we have avoided the use of Oracle-specific SQL in our code.

The tool does not currently support import/export of hierarchical data- specifically, hierarchical XML, where attributes are nested within other attributes. (By contrast, "rectangular" or "flat" XML without nested structures is readily managed through the OLE DB interface, without even needing to write any XML-specific code.) An example of hierarchical XML is that used for interchange by the Gene Ontology consortium, http://www.geneontology.org/ Hierarchical data formats, while somewhat more compact than the equivalent flat formats, are double-edged: their appropriate use requires careful judgment, and is justified only when the relationship between the nested information and the outer block of data that contains it is strictly many-to-one. For such data, however, transfer to/from hierarchical format would be highly desirable. The algorithm for export into such a format is fairly straightforward, and we should be implementing it shortly after enhancing the mapping metadata to track the classes whose objects are to be nested within objects belonging to other classes. Import from a hierarchical XML structure, on the other hand, requires using the XML Document Object Model (DOM). Robust validation, in terms of reporting useful diagnostics if there is an error in the hierarchical data, will also be a relatively involved task.

Acknowledgments: This work has been supported by NIH grants U01 ES10867 from the National Institute of Environmental Health Sciences (Nadkarni) and R01 DC03972 from the National Institute of Mental Health (Marenco). The principal investigator for the SenseLab project is Dr. Gordon Shepherd of the Section of Neurobiology at Yale Medical School.

REFERENCES

1. Huff SM, Haug DJ, Stevens LE, Dupont CC, Pryor TA. HELP the next generation: a new client-server architecture. In: Proc. 18th Symposium on Computer Applications in Medical Care; 1994; Washington, D. C.: IEEE Computer Press, Los Alamitos, CA; 1994. p. 271-275.
2. Friedman C, Hripcsak G, Johnson S, Cimino J, Clayton P. A Generalized Relational Schema for an Integrated Clinical Patient Database. In: Proc. 14th Symposium on Computer Applications in Medical Care; 1990; Washington, D. C.: IEEE Computer Press, Los Alamitos, CA; 1990. p. 335-339.
3. Nadkarni PM, Brandt C, Frawley S, Sayward F, Einbinder R, Zelterman D, et al. Managing attribute-value clinical trials data using the ACT/DB client-server database system. Journal of the American Medical Informatics Association 1998; 5(2): 139-151.
4. Nadkarni PM, Marenco L, Chen R, Skoufos E, Shepherd G, Miller P. Organization of Heterogeneous Scientific Data Using the EAV/CR Representation. Journal of the American Medical Informatics Association 1999; 6(6): 478-93.
5. Shepherd G, Mirsky JS, Healy MD, Singer MS, Skoufos E, Hines MS, et al. The Human Brain Project: Neuroinformatics tools for integrating, searching and modeling multidisciplinary neuroscience data. Trends in Neurosciences 1998; 21(11): 460-8.
6. Nadkarni PM, Brandt CA, Marenco L. WebEAV: Automatic Metadata-driven Generation of Web Interfaces to Entity-Attribute-Value Databases. Journal of the American Medical Informatics Association 2000; 7(7): 343-356.
7. Slezak T, Wagner M, Yeh M, Ashworth L, Nelson D, Ow D, et al. A Database System for Constructing, Integrating, and Displaying Physical Maps of Chromosome 19. In: Hunter L, Shriver BD, editors. Proceedings of the 28th Hawaii International Conference on System Sciences; 1995; Wialea, Hawaii: IEEE Computer Society Press, Los Alamitos, CA; 1995. p. 14-23.