Cruella
Developing a Scalable Tissue Microarray Data Management System

James D. Cowan, BS; David L. Rimm, MD, PhD; David P. Tuck, MD

- **Context.**—Compared with DNA microarray technology, relatively little information is available concerning the special requirements, design influences, and implementation strategies of data systems for tissue microarray technology. These issues include the requirement to accommodate new and different data elements for each new project as well as the need to interact with pre-existing models for clinical, biological, and specimen-related data.

  **Objective.**—To design and implement a flexible, scalable tissue microarray data storage and management system that could accommodate information regarding different disease types and different clinical investigators, and different clinical investigation questions, all of which could potentially contribute unforeseen data types that require dynamic integration with existing data.

  **Design.**—The unpredictability of the data elements combined with the novelty of automated analysis algorithms and controlled vocabulary standards in this area require flexible designs and practical decisions. Our design includes a custom Java-based persistence layer to mediate and facilitate interaction with an object-relational database model and a novel database schema. User interaction is provided through a Java Servlet-based Web interface.

  **Results.**—Cruella has become an indispensable resource and is used by dozens of researchers every day. The system stores millions of experimental values covering more than 300 biological markers and more than 30 disease types. The experimental data are merged with clinical data that has been aggregated from multiple sources and is available to the researchers for management, analysis, and export.

  **Conclusion.**—Cruella addresses many of the special considerations for managing tissue microarray experimental data and the associated clinical information. A metadata-driven approach provides a practical solution to many of the unique issues inherent in tissue microarray research, and allows relatively straightforward interoperability with and accommodation of new data models.

(Arch Pathol Lab Med. 2006;130:817–822)

---

Accepted for publication January 13, 2006.

From the Department of Pathology, Yale University School of Medicine, New Haven, Conn.

The authors have no relevant financial interest in the products or companies described in this article.

Reprints: James D. Cowan, BS, Yale University School of Medicine, Core Tissue Microarray Facility, 310 Cedar St, PO Box 208023, New Haven, CT 06520-8023 (e-mail: james.cowan@yale.edu).

---

Tissue microarrays are a method for analysis of multiple tissue samples on the same slide, with numerous advantages over conventional single-section histology slides.1 Since the invention of machines for the rapid production of this technology, tissue microarrays have emerged as an invaluable tool for performing high-throughput experiments on formalin-fixed, paraffin-embedded tissue samples.2 As with other high-throughput tools such as DNA arrays, tissue microarray experiments can result in large quantities of experimental data. The challenge of storing and managing tissue microarray data differs greatly from that of DNA arrays, however. In DNA microarray technologies, each spot on an array represents a unique cloned cDNA or oligonucleotide, but on a tissue microarray, each spot consists of a small disc of tissue. Generally speaking, each specimen on a tissue microarray will come from a unique individual, allowing you to perform controlled experiments across a large cohort of individuals simultaneously. To take full advantage of a tissue microarray, it is necessary to store and link the clinical information related to each tissue sample placed in the arrays with data read from each array slide.

Recently, many groups have recognized the value of automated quantitative analysis of tissue microarrays as opposed to the traditional method of having a pathologist score the arrays by eye.3,4 One of the benefits of automated analysis is the ability to generate significantly more information on each tissue spot on the array. Although a pathologist will generally assign a limited set of discrete values for a given spot on an array, automated analysis can result in a wide range of continuous values for a given spot, such as the total percentage of observed target in a given area, the ratio of a target in a given area to the target in the entire spot, or the ratio of target in one area to another. As the analytical algorithms and throughput tools advance, the ability to extract more types of data points at a faster rate increases.

For all the benefits of automated analysis, there are an equal number of challenges in storing, managing, and analyzing the resultant data. A system must be constructed that can incorporate new, diverse types of experimental data without requiring wholesale changes to the existing database or user interfaces. The system should be able to provide a link between the experimental data and the myriad of clinical parameters that may be of interest to the researchers. These clinical parameters may also be dynamic and heterogeneous in nature and vary widely, depending on the type of tissue in question. The system...
should be able to scale and adapt as new clinical information becomes available, again without necessitating major changes to the underlying schema. This article describes the implementation of a tissue microarray data management system, nicknamed Cruella, which is based on a hybrid entity-attribute-value (EAV) relational data schema, using object-relational mapping techniques, a custom, Java-based persistence layer to encapsulate the complexities of database interactions, and with a Web interface using an open-source JSP framework (Figure 1).5 The system accommodates tissue microarray metadata, tissue microarray experimental data, and the clinical information associated with each spot on an array and provides a direct link between the experimental and clinical data.

ARCHITECTURE

Backend

The primary influences in designing a database schema for Cruella were the need to store clinical data from diverse and unpredictable data sources, the diverse requirements for different tissue types, and the need to accommodate experimental data values that could represent any potential set of experimental variables. It was impossible to predict what sources of clinical information would need to be incorporated into our system and what parameters this data may require. Also, the automated analysis algorithms were novel and the amount and types of data that they generated could change from one experiment to the next. In brief, it was impossible to predict the types of input that the system would be required to handle. On the other hand, many of the data elements were clear or were the object of well-advanced ongoing efforts in standardization and harmonization.

For the first set of data elements, it became obvious that using a traditional relational schema would require constant modification to the table structures as new clinical and experimental attributes were introduced. These table modifications would result in tables that were excessive in size and were populated with rows of data that consisted primarily of null values. One widely used method for more efficiently managing these types of heterogeneous data is a modeling approach called EAV modeling, also known as row modeling.6-8 Whereas a conventional rela-
tional schema dictated that any attribute for any given element would be defined as a column in the table defining that element, the EAV structure uses a table that stores a reference to an entity, a reference to an attribute, and a value for that entity/attribute combination. (For illustrative purposes, if you had a customer database, “customer” would be an entity; “first.name,” “zip.code,” “e-mail” may be examples of attributes; and “John,” “90210,” “john@domain.com” would be values for the respective attributes.) The element reference is treated just like any other foreign reference. The attribute reference is also treated as a foreign reference, but in the EAV model, it is a lookup in a metadata table that defines the attribute. The value is simply the data point of interest for that particular element. One of the primary benefits of the EAV approach is that you only store values for attributes that are relevant to an entity record. Also, new attributes can be added without requiring changes to the schema or any code that accesses the tables.

Although the EAV structure is efficient and flexible in storing heterogeneous data, queries against the EAV tables potentially can be significantly less efficient than with a conventional schema. The various factors that influence the performance of the EAV schema were explored in detail by Chen et al. The strategy that we developed to maximize efficiency was to employ a hybrid EAV-relational structure that uses conventional relational tables to store the data that conforms well to the relational model, and uses the EAV schema to store the heterogeneous clinical and experimental data. We also provided a straightforward way to migrate specific attributes from one representation to the other, as the design stabilized and there was a desire or need to improve performance of retrieval. We refer to these 2 representations as dynamic and static design elements in the object model.

In the clinical context, there is some information that is stored for all patients and clinical events as well as some information that is relevant to all tissue specimens. These elements are unlikely to change and can be represented by a table object model. This static information may include demographic data, such as name and date of birth; event data, such as age at diagnosis and date last seen; and specimen data, such as date of surgery and surgical diagnosis. It makes sense to store these types of data in a normalized relational format to maximize the efficiency of queries against these tables, and because the object-relational mapping is unlikely to require modification during the lifetime of any applications built around them. In our hybrid design, these normalized tables can also serve as entities in the EAV schema. As an example, we have defined a clinical_case table in which we store clinical event data. The EAV table clinical_case_element uses the primary key from the clinical_case table as the entity reference and the primary key from the clinical_attribute table as the attribute reference (Figure 2). Similarly, we have a table called spot that stores static information about a spot in a tissue microarray, such as row/column position. We use the primary key from the spot table as the entity reference in our score_element table, which stores individual score values from experiments (in the case of score_element, the entity is actually a combination of the spot primary key and a primary key from the table that defines the experiment or assay). In both of these examples, the attribute is defined by a foreign reference to the metadata table. As a note, although the framework was designed to be portable across database platforms, we have employed MySQL (MySQL AB, Sweden) as our reference relational database management system.

Middleware

The queries necessary to manipulate data in the hybrid EAV-relational schema can be complex. To encapsulate and manage this complexity, we employed object-relational mapping and object-oriented data persistence strategies that provide a robust, flexible middleware layer with a lightweight application program interface (Figure 3). This multilayered layer consists of an object hierarchy that provides services such as connection management, dynamic query generation and execution, optimistic concurrency management, maintenance of object relationships, and data persistence. The object hierarchy consists of several objects for each mapping that collaborates to manage all aspects of persistence:

- the “support” objects, which handle all database interactions at the lowest level using the database driver, in our case Java Database Connectivity (JDBC);
- the “home” objects, which maintain the relationship of the higher level objects to the mapped table, essentially populating the “entity” objects with the queried data from the database;
- the “entity” objects act as data stores and manage all operations on the persisted data;
- the “façade” objects are lightweight pointers to the persisted entity data and provide the public accessor methods that allow access to the persisted data on demand, and provide caching of queried data to improve performance for repeated access.

The manner in which these objects function and interact differs depending on whether they map to a conventional table or an EAV structure. In either case, the public methods used to retrieve data are similar, although the programmer using the framework must take into account whether they are operating with a dynamic or static mapping. If the mapping is dynamic, then the entity object provides methods for retrieving or altering one or more
attributes for a given entity. The benefit of encapsulating the logic for the core queries that are needed for these retrievals and manipulations remains consistent, regardless of the underlying database schema. Adding a new attribute for an entity is minimized to a simple method call on the related entity object with the attribute name and the value passed in as arguments in the same way that a simple setter method would be provided for a static field in a traditional mapping, again with the value being passed in as an argument. In all cases, the lower level objects collude to enforce referential integrity, manage concurrency, and avoid duplications.

One of the most pertinent issues that arises with this type of object-relational mapping strategy is change management. Although the number of Java classes required for each entity is relatively small, when the domain space is large, it amounts to a significant code base. In our current implementation, there are 156 domain-specific persistence classes with more than 20 000 lines of code. Needless to say, maintaining this code base during periods of refactoring would be overwhelming.

To address this issue, we created a code-generation system that is driven by the popular Apache Ant build tool and the Jakarta Velocity template engine (Apache, Forest Hill, Md). This system takes a group of properties files, one for each entity (or mapping), and generates the entire persistence object hierarchy, including the embedded SQL statements in the support objects. The properties files indicate whether attributes should be treated as EAV attributes or conventional database fields, and the template system will take this into account when generating the Java code and the necessary SQL. This system allows us to quickly regenerate the persistence layer when changes are made to the database schema. Also, when it is necessary to modify the underlying logic, it is not necessary to propagate those changes for each entity by hand. We simply make the changes in the templates and regenerate the classes. We can also maintain separate templates for different database management system platforms such that migrating from MySQL to Oracle (Redwood Shores, Calif) or Sybase (Dublin, Calif) requires only regenerating the persistence classes. A side benefit of this system is that it is easy to apply to any domain space. Implementing a complete object-relational mapping and persistence layer for a project is simply a matter of creating the properties files for the mappings and running the generation process. In fact, systems with different domains (GeneCube, a gene array database application, and YTMA Manager, our array inventory and slide management utility) have been developed by our group with the identical persistence layer mechanism but differing domain objects.

**Front End**

The original user interface implementation for Cruella was a stand-alone client application written in the Java programming language. Developmentally speaking, this provided a powerful, flexible platform for allowing users to manage their experimental data and to provide controlled access to the related clinical parameters of interest to them. As the number of users increased, and the burden of deploying patches and upgrades to the software became too great, the decision was made to migrate to a Web-based front end.

Among the available, open-source, Java-based, Web-application frameworks, we chose to use SOFIA (the Salmon Open Framework for Internet Applications). An open-source project sponsored by Salmon, LLC (Mineola, NY), SOFIA includes a robust set of tools and a Web application framework based on JavaServer Pages (JSP) and Java Servlet technologies. Although SOFIA is similar to the Apache Struts framework and other J2EE Model-View-Controller (MVC) architectures, it encompasses prebuilt user interface components and tools that integrate with popular development environments like Dreamweaver, IntelliJ, and Eclipse. This integration allows for rapid application development as well as robust testing and debugging mechanisms.

The SOFIA is based on the MVC architecture. The MVC architecture separates an application into 3 loosely coupled components: the data access component (the model),
the presentation or user interface component (the view), and the business logic component (the controller). In SOFIA, the view is implemented using JSP and a vast collection of custom tag libraries. One or more controllers is defined in each JSP page, each of which maps to a Java class. These classes provide the controller layer of the MVC architecture. The SOFIA allows any Java object to be used as a data model as long as it provides public accessor methods. Although SOFIA includes tools for generating models using a database query builder, we found that we were able to use the façade classes from our persistence layer directly, without modification.

**COMMENT**

The development of Cruella was guided by 3 essential requirements: (1) to provide robust storage for experimental data; (2) to provide linking between experimental data and the associated clinical information; and (3) to provide tools to manage experimental information. After the system architecture was formalized, we decomposed the user requirements into a workflow based on tasks. The users would need interfaces to put data into the system, get data out of the system, and generally manage their data as they saw fit. We decided that clinical information and array annotation would be handled administratively. Also, because we were primarily interested in automated analysis, we were not concerned about storing or managing individual spot images (although the database is designed to accommodate image storage, we did not develop it into the user interface).

**Defining Arrays**

To define an array in Cruella, we start by incorporating the available clinical information for each tissue specimen into the clinical tables in the database. The key entity in the clinical structure is the specimen. As an array is constructed, the position of each specimen on the array is recorded in a spot table, which is in turn linked back to the corresponding specimen record. This ultimately provides the bridge between the experimental and clinical data. The array is assigned a permanent, unique identifier known as an YTMA number, which is used both institutionally and by Cruella users to refer to that array.

**Importing**

Once an array is defined in Cruella, users are able to begin putting experimental data into the system (a process we refer to as importing). Importing data requires the user to follow a sequence of steps to select the appropriate array, define the experiment, and provide a tab-delimited text file containing their experimental data. The experiment file is formatted such that each row contains information for one spot on the array, with the top row acting as a “header” row. Two columns are required to indicate the position of the spot on the array ("row" and "column"). These columns are used to link the experimental value to the spot record provided during the array definition. The file can contain one or more experimental data columns and there is no limit on the number of experiment parameters that a user can provide. Occasionally, we see files imported with a dozen or more parameters.

**Exporting**

As soon as a user’s experiment is imported, it is made available for export with clinical data. Again, the user interface walks the user through a sequence of steps to select the desired array, select the desired clinical parameters (Figure 2), and finally to select one or more experiments to include in the export. An export file is generated and made available to the user for download. By default, the export file is formatted as a tab-delimited text file where the first row acts as a header row to identify the contents of the respective columns. This format is compatible with most popular spreadsheet programs as well as most statistical applications. Because the export is metadata driven, we are able to easily control the types of data that are included in an export (for instance, we may not want to include hormone replacement therapy data for an export of an assay on colon cases). This metadata approach also allows us to incorporate new export formats by simply extending an abstract export class and defining in the metadata tables the desired field mappings. The default exporter, which organizes the data into tabular format in which the fields are mapped to column headers, has met the needs of virtually all users we have encountered to date. However, by mapping the fields to defined data elements, and using XML packages now intrinsic to the Java programming language, we can easily provide an adapter to organize the data in any specified XML format, such as the common data elements described by the tissue microarray data exchange specifications developed by the Association for Pathology Informatics.

**Management**

Cruella also provides users with simple management tools for their experiments. They can include a range of parameters to document each assay such as protocol data, references to laboratory notebooks, slide numbers, and quality fields that assist in organizing experiments. They can also retrieve information about the arrays themselves, such as total number of cases or number of total control spots, or they can view and download the exact layout of the array in an Excel spreadsheet.

**Alternative Designs**

Since the introduction of tissue microarray technology, a small number of articles describing systems for managing tissue microarray data have been published. These systems range in complexity from spreadsheet applications to larger-scale relational database systems with custom user interfaces. Although, for the most part, these systems are well implemented and seem to serve the needs of the researchers who developed them, we found that the designs did not adequately address many of the issues that we were facing. Namely, most of the systems were centered on storing images and facilitating the manual analysis of the individual spots. Because these were not requirements for us, we did not focus our development in this direction. Also, many of the systems did not directly address long-term storage and management, storing and linking clinical data for multiple tissue types, and how to accommodate new sources of experimental data from automated analysis.

**Other Developments**

Although Cruella has proven to be efficient and robust in managing the large amount of data that are generated by our users, there are a few developments in the works to improve and extend the functionality of the application. Primary among these developments is the integration of...
radio frequency identification technologies into Cruella for the purposes of tracking and managing not only our array block inventory, but also our donor tissue blocks. Using radio frequency identification tags embedded in the paraffin blocks, we are able to rapidly identify a tissue sample and retrieve the clinical information that we store for that patient.

Currently, we have a pilot project designed to evaluate this radio frequency identification integration into Cruella and we have tagged a significant number of tissue samples for this purpose. We are able to use this technology as we construct new arrays by scanning the blocks as they are cored, and recording the position of the core automatically. This has the potential to minimize errors related to manual data entry of this information. It also helps to ensure that we are correctly identifying a tissue sample. Although there are many issues that need to be addressed, our preliminary experience has been positive, allowing rapid and accurate array map construction.

CONCLUSION

The combination of a hybrid EAV/conventional database schema along with object-relational mapping and persistence strategies as described in this article has proven to be a good foundation on which to build our tissue microarray system. We have been able to leverage normalized table structures to accommodate the stable portions of our data model, and use EAV structures for areas of heterogeneity and instability, of which little can be known about the data attributes in advance. Although the hybrid approach is conceptually complex, the complexities are encapsulated by a robust middleware layer incorporating standard object-relational mapping techniques and a set of robust persistence classes. The result is a highly scalable, flexible data repository and management system that is capable of meeting the needs of its users now and into the foreseeable future.

Cruella has been in use since February 2002 and, although the user interface has undergone continual change, the system architecture has been stable. The most significant schema changes have been the migration of some clinical attributes from dynamic to static as they have stabilized. Cruella now has approximately 30 users and stores almost 800 individual experiments. These experiments account for more than 1675,000 unique data points, with more added every day. There are currently 15 different types of tissue represented on 35 tissue microarrays with 3400 specimen records from 3200 unique patients.

This work was supported by grants POL HL 633571 and T15 LM07056 from the National Institutes of Health, Bethesda, Md (Dr Tuck). The work was also supported by the Breast Cancer Alliance of Greenwich, Conn, grant NCI R21 CA100825 from the National Institutes of Health, and a Yale Comprehensive Cancer Center Shared Resource grant (Dr Rimm).

References