TDAN.com
The Data Administration Newsletter
Since 1997

A Framework for Logical Data Models in Courts

by Derek Coursen, James E. McMillan
Published: December 1, 2010

(Article URL: http://www.tdan.com/view-articles/14749)

Success in implementing a court automation system depends on developing a high-quality logical data model. Certain patterns regarding representation of data on actors in the judicial process, cases, component matters (charges and civil claims), and events and tasks are generically applicable to any court situation. These patterns offer a framework for software development teams and non-technical stakeholders to communicate more effectively about the implications of key data modeling decisions in courts.
Success in building a court automation system depends on first arriving at a high-quality logical data model. As in any software project, the data modeling phase is the one that most critically affects the final quality of the system—including its costs, flexibility, and ability to meet user requirements and be integrated with other systems (Moody & Shanks, 2003). Unfortunately, teams in court automation projects may narrowly focus on defining functional requirements based on judicial work processes, either as they currently exist or as they are being reengineered; the team may then merely infer a data model that will support those immediately knowable requirements, with the result being less comprehensive and flexible than would be optimal.

There is, however, an alternative approach: insights based on data modeling as a discipline in its own right can lead to a broader and more complete perspective on business requirements, and can produce a software system that is more flexible, better at meeting unanticipated user needs, and less expensive in the long run. This has high value in a judicial environment: although court systems may seem slow to change, in fact they are unusually dynamic in their organizational structures and business processes. Both because of annual legislative changes and because of courts' interest in testing new judicial solutions, judicial data models should be designed to support change.

A powerful way to jump-start the data modeling process and reinforce its quality is to consult data model patterns. Just as architects refer to the blueprints of existing buildings while designing a new one, David C. Hay's (1996) work showed that widely applicable patterns can guide data modelers to understand the complexities of specific subject areas and the implications of choices they will confront. Hay and others (e.g. Silverston, 2001a, 2001b) have produced patterns useful in a multitude of common business situations.

There has so far been little work on specific issues around logical data models for court automation systems, though. The gap is probably the result of the fact that so much of the court technology field's recent attention has focused on development of a common exchange standard, the Global Justice XML Data Model (GJXDM) (http://www.it.ojp.gov/jxdm/). But while useful for data integration projects, such standards are limited in their ability to describe the meaning of the underlying data (Hay, July 2007). In fact, it can be necessary to reverse engineer an XML standard in order to examine its assumptions and how it might be mapped to a logical data model, as Hay has done with the GJXDM (April 2007, in press).

This article aims to fill that gap by proposing data model patterns that work well in judicial environments in general. The authors arrived at these patterns in the following way: In January 2005, after several years of discussing the idea of universal court data model patterns, Jim McMillan was leading the development of a new court automation system for Bosnia-Herzegovina (BiH) and invited Derek Coursen to develop a model that could serve both BiH and any other jurisdiction. Working with Bosnian court officials and data managers, the authors spent a week white-boarding a model. The sessions followed a set of rules that are heterodox by the standards of most software development projects: (1) refer when possible to published data model patterns; (2) focus on entities, relationships and attributes, discussing work processes only insofar as they directly affect those data model components; (3) focus on the BiH courts first but only as a springboard to broader generalizations; (4) focus on the current BiH reality first but only as a springboard to discussing potential needs for flexibility; (5) support eventual physical implementation by seeking a level of generalization that is a happy medium between inflexible specificity and unwieldy theory.

The resulting data model was subsequently implemented in a system that has been rolled out throughout Bosnia-Herzegovina. The system implementation there encompasses more than 3,000 users in more than 70 locations. It has been extended beyond the trial court to the appellate courts and is currently being extended to prosecutors. The system is now being piloted in the courts of Indonesia as well. This approach has paid significant dividends in efficient system implementation for both countries' judicial systems. In the court automation world, system transfer projects have failed time and again simply because the fundamental database design was not flexible enough to map the individual court organization and processes. In contrast, these implementations are being done not by rewriting the database structures but rather by creating appropriate work and task/role templates that fit within the flexible data model. Transferring the system from BiH to Indonesia took less than six months after technical training of local staff.

This article presents the major patterns and decision points that informed the model. It is intended to offer a set of considerations that will help software development teams and non-technical stakeholders communicate more effectively about the implications of key data modeling decisions in courts. While the areas outlined herein do not constitute a complete data model, they address the most important concerns regarding the four most central regions of any court data model: actors in the judicial process; cases; component matters (charges and civil claims); and events and tasks.

Actors in the Judicial Process

Business enterprises typically deal with various kinds of people and organizations, and a data model for the enterprise should begin by accurately representing all of those kinds. This is certainly true for judicial case management systems, where it is necessary to track all the different players in legal matters (Steelman, Goerdt & McMillan, 2000).

To do this, the authors designed a generalization hierarchy that essentially follows Hay's (1996) approach. It flexibly provides a location in the database for representing any person or organization of any kind. This is shown in Figure 1, where there are three levels of hierarchy. The top level is ACTOR. (This term was chosen over "party" because the latter has a particular narrow technical meaning in legal cases.) An ACTOR may be one of two subtypes, either a PERSON or an ORGANIZATION. Certain kinds of PERSON and ORGANIZATION—for example, ATTORNEY, POLICE AGENCY and COURT—may in turn be represented by subtypes in the third level below. These third-level subtypes are necessary for situations where the subtype of PERSON or ORGANIZATION has attributes specific to it that are not applicable to persons or organizations in general. They are also necessary when a relationship uniquely exists between that particular subtype and another entity (Reingruber & Gregory, 1994).

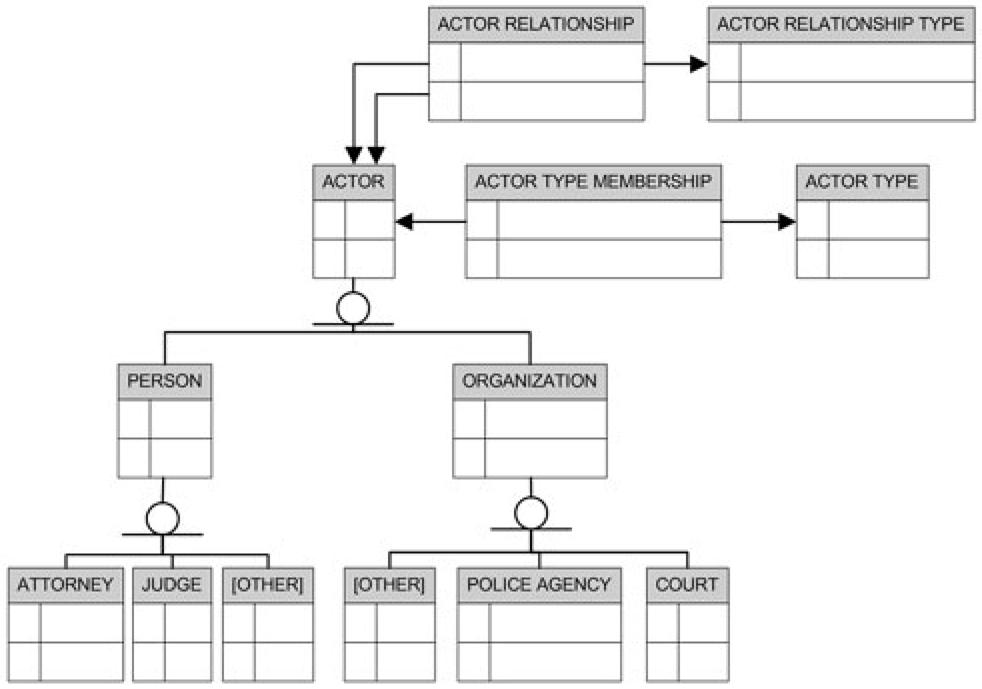


Figure 1: Actors in the Judicial Environment

One advantage of a generalization hierarchy is that for many purposes, the entities at the upper levels of the hierarchy suffice. Certain types of people and organizations may exist in the judicial environment and need to be recorded in the database, yet—according to the criteria above—need no subtype entity in the data model. For example, there might be a need to record interactions with certain government agencies, but the ORGANIZATION entity might encompass all the required data without any need for a GOVERNMENT AGENCY subtype entity. At the same time, though, the system should be able to record that fact that organization x is in fact a government agency. For this purpose, the best practice—again following guidance articulated by Hay and others—is to have an entity ACTOR TYPE that lists all of the kinds of persons and organizations of interest to the enterprise. The same actor may belong to one or many actor type, either simultaneously or over time, and the facts of who belongs to which types are then represented in the entity ACTOR TYPE MEMBERSHIP.

In order to apply this three-level scheme to a particular judicial environment, analysts will need to make an inventory of all of the kinds of people and organizations that the system may need to record, and then—based on the criteria above—decide which of them call for a subtype entity. On the PERSON side, the JUDGE is usually a key subtype, as a great deal of the system's purpose is to support and streamline judges' work. An ATTORNEY might require a subtype entity because of the need to represent bar membership information. POLICE OFFICER often will also, due to the need to associate the officer with other entities (e.g. an EVIDENCE ITEM), or to represent facts such as the officer's date of graduation from the police academy. On the ORGANIZATION side, the most important third-level subtype will usually be COURT. It is possible to imagine an automation system that would not need this entity (if it were built for a single court that never interacted with any other court) but the scenario is unlikely. Judicial systems generally involve multiple courts representing different levels of responsibility, different geographic areas, and different specialized tasks. Other commonly occurring subtypes of ORGANIZATION are POLICE AGENCY and CORPORATION.

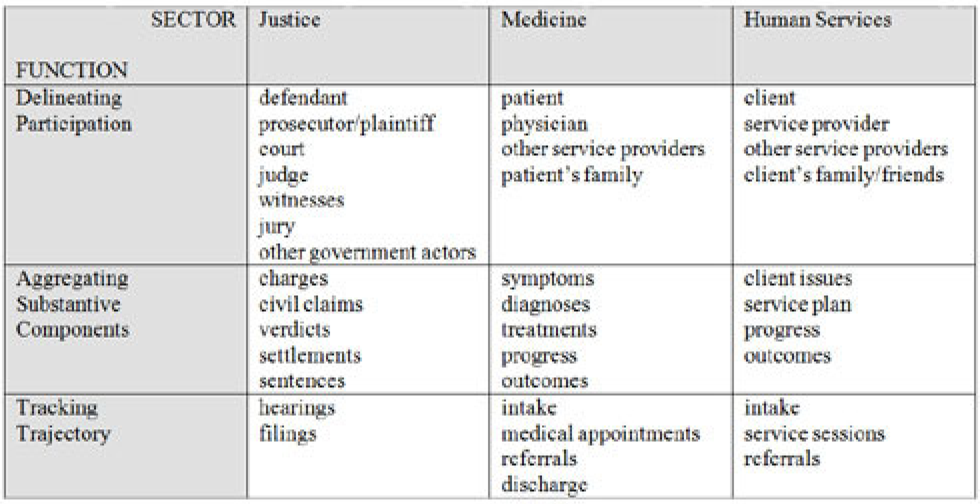
Inclusion in the ACTOR TYPE list is only appropriate for types of person and organization that, for their defined identity, are independent of other actors. For example, a spouse is only a spouse in relation to another spouse; spouse would therefore not be an appropriate ACTOR TYPE. Instead, the data model includes two other entities, ACTOR RELATIONSHIP TYPE and ACTOR RELATIONSHIP, which represent the ways that different actors may be linked together (e.g. by marriage) in the database. Similarly, an ACTOR TYPE must also be based on characteristics that are independent of particular cases. A witness, for example, is only a witness within a specific case. Thus, the concept of witnesshood must therefore be represented elsewhere in the data model in a way that depends upon the case.

The flexible structure outline above is useful for courts in many ways. For example, over the course of a career a practicing private attorney will have many interactions within the local court. The attorney will most commonly represent his or her clients in their cases. But an attorney who is not paid can easily become a plaintiff in a matter against that same client. Further, an attorney might be sued for malpractice if the client is not satisfied with their representation. And that same attorney can also be called by the court to serve as a mediator or trustee for another set of cases. Later, the attorney might become a judge and preside over other cases. As a private citizen, even, that attorney might appear as the witness, victim, defendant or jury member for yet other cases. In short, a single person may have multiple complex ways of relating to the judicial system. This structure allows a court automation system to represent that person in a single unified record.

Cases

Courts deal with cases, but what that means is not obvious. The term "case" is not exclusive to the law, but appears in medical and social service settings as well. It originates, apparently, in the physical folders that doctors and lawyers traditionally used to organize their paperwork. It has been argued that the case is therefore "an egregious transfer of an analog concept to the digital environment" (Fitch, 2007). In comparison with entities that are more directly perceptible in the business environment, the notion of a case does seem oddly arbitrary and socially constructed—both qualities that raise red flags for data modeling.

It is helpful to unpack the structural and functional meaning of a case. Though different professional settings have different practices and terms of art, the case turns out to fulfill the same role in all of them. First, it delineates participation; second, it aggregates smaller substantive components; and third, it tracks trajectory. These commonalties and how they apply to different professions are shown in Figure 2:



SECTOR FUNCTION	Justice	Medicine	Human Services
Delineating Participation	defendant prosecutor/plaintiff court judge witnesses jury other government actors	patient physician other service providers patient's family	client service provider other service providers client's family/friends
Aggregating Substantive Components	charges civil claims verdicts settlements sentences	symptoms diagnoses treatments progress outcomes	client issues service plan progress outcomes
Tracking Trajectory	hearings filings	intake medical appointments referrals discharge	intake service sessions referrals

Figure 2: The Functions of the Concept of "Case"

For the purpose of information system design, though, the case still needs to justify itself. Would it not be cleaner to model these aspects without using the notion of a case? Some patterns for human service data models do exactly that, dealing with trajectory (Coursen & Ferns, 2004) and with participation and components (Coursen, 2006) in ways that make the case an unnecessary concept.

It would be far more difficult, though, to abolish the case in a court data model. A judicial system has a particular need to be able to effectively pass complex packages of component matters from one court to another, and it must satisfy an extremely high level of public accountability for keeping records on its proceedings and the disposition of all matters. For managing these requirements, the case is irreplaceable.

Two interrelated questions are a good starting point for thinking about how to represent cases in a data model: how do cases relate to courts, and how do cases relate to other cases? These questions helpfully bring out conflicting uses of the term.

A newspaper might report that "case x was decided in the plaintiff's favor in District Court, but it then went up to the Appeals Court and the decision was overturned." This journalistic sentence implicitly asserts that case x remains case x when it moves from court a to court b. From a bird's eye view of the judicial system, this makes a certain amount of sense (even though the subject matters of the original and appealed cases will often be different in that the first may focus on the facts while the second focuses on the law). In the realm of the two courts' record keeping, though, are the two cases the same or even linked? In paper-based environments, or when the two courts have different information systems, it is almost certain that the two courts will maintain different case files (and different case numbers). And if the Appeals Court were to rule on the point of law, and were then to send the case back to District Court for further work—would the second passage of the case through District Court entail creating a new case record or not?

For an integrated information system that spans multiple levels of a judicial system, a basic design goal should be for the data model to both preserve the continuity of the case across different courts and also to manage each passage of the case through each court as a discrete piece of work.

The notion of "continuity of a case" is strictly applicable, however, only if a case has an unchanging identity as it moves from court to court. In many jurisdictions that is not necessarily true. A judge may split a single case into multiple cases (e.g. when there are multiple defendants with conflicting interests). Conversely, a judge may consolidate multiple cases into one (e.g. when multiple plaintiffs sue the same defendant for the same claims, or when it is more efficient to try a defendant once for offenses that occurred, and were originally charged, over a period of time). Given these possibilities, the design goal becomes more complex: the data model must also accurately represent how cases may combine or divide due to these contingencies.

To do all this in a complete and flexible way, it makes sense to represent how cases relate to each other using a structure entity CASE ANTECEDENCE. Using this model as shown in Figure 3, each CASE may be the antecedent of one or more subsequent cases, and each case may also have multiple antecedents. This allows the model to represent how a case may flow up and down a judicial system and how it may split or be consolidated with other cases.

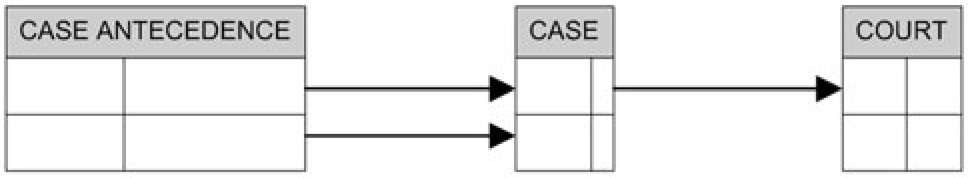


Figure 3: Cases Related to Their Antecedent Cases

The COURT is, of course, a subtype of ACTOR, but it makes sense for the COURT to be related directly to the CASE since the data model is asserting that each case belongs to one court (and the transfer of a case to a different court requires the creation of a new case record).

Cases, of course, involve many actors. For the purpose of recording their participation, a flexible structure is important. The associative entity CASE ROLE, shown in Figure 4 below, allows many actors to be involved in many different cases. The list embodied in CASE ROLE TYPE will include (at least) judge, defendant, plaintiff, prosecutor, defense attorney and witness.

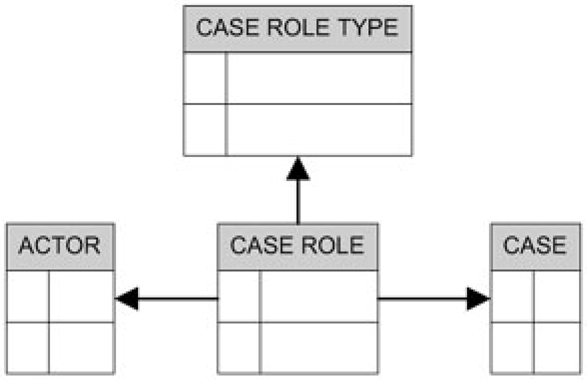


Figure 4: Actors Involved in Roles in Multiple Cases

According to this structure, a single case can have more than one defendant. As will be discussed below, this has important implications for some other regions of the data model. It forces them to refer to CASE ROLE, rather than merely CASE, in order to link any other entities to a particular defendant. That may seem like an unnecessary complication in jurisdictions whose practices stipulate that a case can have only one defendant. Such practices, however, are potentially subject to change, and it is prudent to have a data model that can accommodate such change in advance. (In fact, the rule of one defendant per case sometimes originated in the limitations of earlier computer systems!)

Component Matters (Charges and Civil Claims)

A legal case may be thought of as composed of multiple smaller matters that will be resolved individually. In a criminal case, for example, the state may charge John Smith with burglary, trespassing and possession of marijuana; in a civil case, a plaintiff may make a tort claim of injury to body and also a claim of damage to property. For the purposes of designing this region of the data model, criminal and civil cases can be treated alike; the discussion below will use criminal cases as an example.

A primitive data model might start from the notion that a CASE can have multiple child CHARGE records. This turns out to be too simplistic, however. The set of charges against a defendant often evolves through the life of a case. That evolution can include the addition of new charges, the dropping of previous ones, and the amending (up- or down-grading) of charges, as when a prosecutor decides to reduce a murder to manslaughter. When such a change occurs, facts that need to be captured include the date; the new charge, changed charge, or disappearance of a previously existing charge; and the reasons for each change. How should the data model represent this evolution?

An added complication is that the stages a case may follow cannot be assumed to be tightly fixed. Different jurisdictions have different procedures; within a given jurisdiction, particular judges may be permitted by law to follow idiosyncratic practices; and the procedures of any jurisdiction are subject to modification originating in administrative, legislative or higher judicial decisions. For all these reasons, .a flexible data model should not embody assumptions that a particular case trajectory is normative. Rather, it should be able to represent the evolution of charges no matter how the case may unfold. This approach is particularly valuable when developing a system that will need to serve a variety of courts at different levels, or across different jurisdictions.

An effective way to do this is to state that each CASE ROLE (representing, in real life, each defendant) may have one or many CHARGE SETs, each of which will have as members one or more CHARGE, each of which belongs to one CHARGE TYPE. These CHARGE SETs, then, will need to relate to other areas of the data model that record the progression of the case.

Here is it useful to introduce the concept of an EVENT, which will be discussed in more detail in the following section. An EVENT can here be defined as a significant (recordable) happening that occurs in legal proceedings, or that is scheduled to occur in the future, and that involves the court officially doing something or recognizing something that someone external has done. An EVENT always has a date, and may in some instances have a time and duration. Each EVENT is an instance of one EVENT TYPE. Common types could include hearings, motions, recognition of having received a deposition, passing of sentence, etc.

Each CHARGE SET, then, represents the set of charges that exists as of the moment of a particular EVENT. This model, shown in Figure 5, flexibly captures the full history of a defendant's charges as they evolve through the life of the case. It is, incidentally, advantageous for situations in which criminal history repositories depend on court information systems for clearing the original charges filed by police agencies.

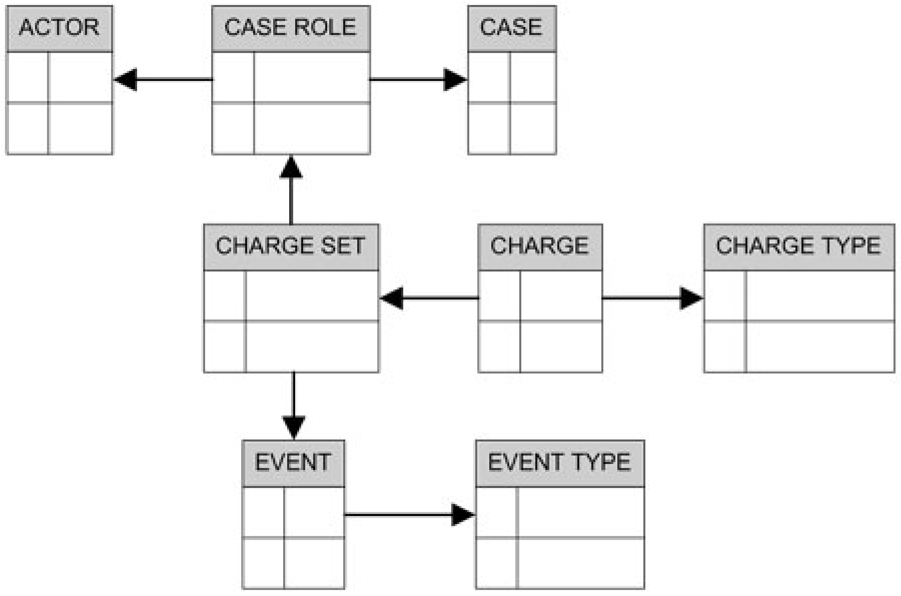


Figure 5: The Evolution of Charges Through the Life of a Case

Events and Tasks

One of the major functions of a court automation system is to manage the court's calendar of cases—often referred to as the schedule or docket—by facilitating the court's workflow around those cases. For the purposes of creating a flexible data model, it is useful to conceptualize the court's work generically as a series of events and tasks. The EVENT has already been defined, above, as a significant happening in which the court officially does something or recognizes something done.

This may seem to be an overly generic conception. Does it make sense to model both a hearing and a motion as types of EVENT? A hearing seems structurally different from a motion or the passing of a sentence. The latter, in fact, often occur within the former. Nonetheless, the concept and entity EVENT can conveniently accommodate all of these possibilities by having a recursive structure such that one EVENT can contain another EVENT.

Certain types of events such as hearings inherently have time and duration, and these attributes are then usable by the system for calendaring functions. An EVENT that has duration will have an attribute indicating its calendaring status, e.g. scheduled, occurred, cancelled. Those types of events which can be calendared may need to be related to a FACILITY entity representing courtrooms and other physical spaces being managed through the calendar.

In relating the EVENT to the major entities described in previous sections, it is important to consider the complexity of court operations. For example, it would be tempting to say that each case may include multiple events. True enough, but each event may also pertain to multiple cases, as when a judge deals with several within a single hearing. This calls, therefore, for a many-to-many relationship where the associative entity CASE EVENT is the linkage between CASE and EVENT.

Similarly, at first glance it might seem that events and actors ought to have a many-to-many relationship. While that is intuitively true, it misses an added dimension: the actor's role in the particular case is such a defining aspect of their participation in the event that the data model should represent the linkage as an EVENT PARTICIPATION between a CASE ROLE and an EVENT (not between an ACTOR and an EVENT).

For the purpose of streamlining workflow, this area of the data model is critically important because it permits the management of tasks. Unlike an EVENT, which has an officially recordable character, a TASK is here defined as simply something that court staff must do. Each TASK is an example of one TASK TYPE. Examples might be scheduling a next hearing, writing up a subpoena, having it served, etc. The TASK will need to contain an attribute representing the person who carried it out. Depending on the workflow practices, it may also make sense for each TASK to be assigned in advance to one employee.

Since the data model includes both events and tasks, it is possible to use it to develop applications that will support a workflow engine. An event may have a certain set of prerequisite tasks which must be completed before it can be scheduled or marked as completed. The scheduling or completion of an event, on the other hand, may spawn a new set of required tasks. In this data model, all such rules could be stored in the entity EVENT TASK RULE.

The systematic collection of data on events and the tasks that they require can enable court administrators to develop sophisticated management statistics. In addition to traditional court caseload status and case event statistics, the data model described here supports real-time measurement of how much effort is demanded by any particular case, or by any category of cases over a period of time. Statistics of this kind are helpful for developing automated random case assignment systems. In addition to estimating how many cases of particular types a judge should be assigned, it is possible to make adjustments for complex or difficult cases that require considerable work to adjudicate (McMillan, September 2005).

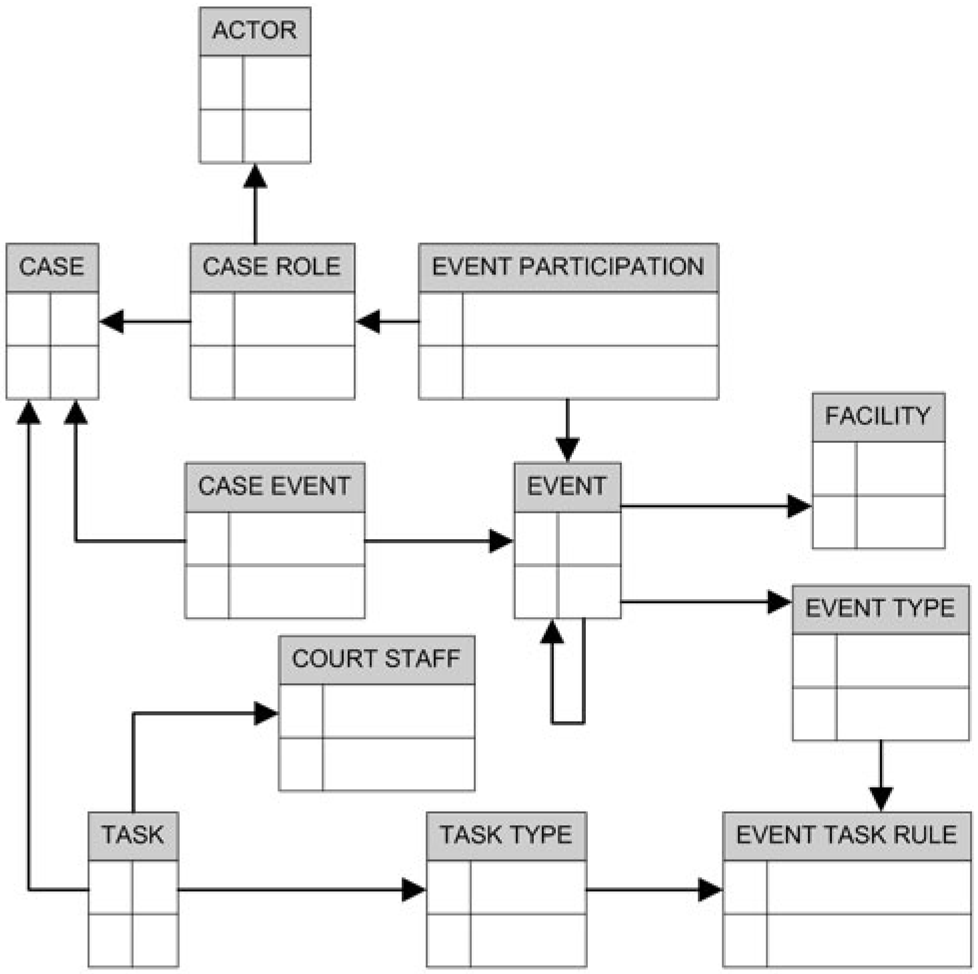


Figure 6: Managing Events and Tasks in the Court

Conclusion

The patterns discussed in this paper are generically applicable to court settings. They do not by any means constitute a complete data model: areas for further work include sentencing, complex categorization of statutes, and financial transactions. These patterns do, however, provide a framework within which application developers and non-technical stakeholders can immediately orient themselves to the major issues that any court data model will confront.

References:

Coursen, D. & Ferns, B. (2004). Modeling participant flows in human service programs. Journal of Technology in Human Services, 22 (4), 55-71.

Coursen, D. (2006). An ecosystems approach to human service database design. Journal of Technology in Human Services, 24 (1), 1-18.

Fitch, D. (2007). Designing Databases Around Decision Making. In M. Cortes & K. Rafter (Eds.), Nonprofits and Technology: Emerging Research for Usable Knowledge (pp 135 – 147). Chicago: Lyceum Press.

Hay, D.C. (1996). Data Model Patterns: Conventions of Thought. New York: Dorset House.

Hay, D.C. (July 2007) “Data Structure: Data Modeling or XML?” Data Administration Newsletter. Downloaded March 2010 from http://www.tdan.com/view-articles/5538.

Hay, D.C. (April 2007) “Global Justice Entity Relationship Model: A Conceptual Entity Relationship Model.”, presentation to the Federal Data Architecture Subcommittee, April 12, 2007. Downloaded March 2010 from
http://epametadata.wik.is/Federal_Data_Architecture_Subcommittee_DAS_Knowledgebase.

Hay, D.C. (In press). Enterprise Model Patterns: Describing the World. Bradley Beach, NJ: Technics Publications.

McMillan, J. E. (September 2005) Status, Events, Weights and Measurement: Court Management Statistics in the BiH CMS. Ninth National Court Technology Conference, Seattle, WA.

Moody and Shanks (2003). Improving the quality of data models: empirical validation of a quality management framework. Information Systems 28, p. 619

Reingruber, M.C., & Gregory, W.W. (1994). The Data Modeling Handbook: A Best-Practices Approach to Building Quality Data Models. New York: John Wiley & Sons.

Silverston, L. (2001a). The Data Model Resource Book. Vol. 1: A Library of Universal Data Models for All Enterprises. New York: John Wiley & Sons.

Silverston, L. (2001b). The Data Model Resource Book. Vol. 2: A Library of Universal Data Models for Specific Industries. New York: John Wiley & Sons.

Steelman, D.C., Goerdt, J.A., & McMillan, J.E. (2000). Caseflow Management: The Heart of Court Management in the New Millennium. Williamsburg, VA: National Center for State Courts.

Go to Current Issue | Go to Issue Archive

Derek Coursen - Derek was formerly Director of Information Management at the Vera Institute of Justice, where he consulted on the design of judicial data models for agencies including the High Judicial and Prosecutorial Council of Bosnia-Herzegovina, the Judicial Information Division of the State of New Mexico, and the Office of the District Attorney of Charlotte-Mecklenburg, North Carolina. He joined Public Health Solutions in 2007, and currently directs informatics around the administration of New York City’s portfolio of HIV care and prevention contracts. He has a particular focus on how information systems for public service settings can be designed to maximize flexibility and to support performance measurement and evaluation as well as operations, and he has previously published articles on advanced practices for data modeling in the human services. He earned M.S. degrees in management (New York University Wagner School of Public Service), information systems (Pace University) and information and library science (Pratt Institute). He may be contacted by email at derekcoursen@hotmail.com or by phone at (347) 401-3649.

James E. McMillan - James joined the National Center for State Courts in 1990 and currently serves as a Principal Court Technology Consultant. McMillan is senior faculty for the Institute for Court Management, and has provided technical assistance for trial and appellate courts and administrative

offices in all 50 states in the USA. Notable consulting projects include the United States Supreme Court, Arkansas, and Massachusetts Supreme Court, and statewide court automation projects with Rhode Island, Maine, New Jersey, South Carolina, and Vermont. He also created the architectural concepts for the Orange County, Florida ICJIS project. Internationally, McMillan has provided expertise to courts in Bosnia & Herzegovina, Bahamas, Croatia, Egypt, Indonesia, Kosovo, Trinidad & Tobago, Serbia, Ukraine, Russia, and the United Nations International Criminal Tribunal. As Director of the Court Technology Laboratory project for eleven years, he was the co-recipient of the Howell Heflin Outstanding Project Award from the State Justice Institute and was a co-founder of Courtroom 21 with the College of William and Mary School of Law. Prior to joining the NCSC, he was the founding Information Technology Director for the Arizona Administrative Office of the Courts. He is co-author of *A Guidebook for Electronic Court Filing* and a contributing author to *Caseflow Management: The Heart of Court Management in the New Millennium*. McMillan received his BA in government from New Mexico State University and an MPA with a specialization in judicial administration from the University of Southern California. He may be contacted by email at jmcmillan@ncsc.org.