COMPUTERS AND BIOMEDICAL RESEARCH **29,** 194–221 (1996)
ARTICLE NO. 0016

Design of a Clinical Event Monitor

GEORGE HRIPCSAK, PAUL D. CLAYTON, ROBERT A. JENDERS, JAMES J. CIMINO, AND STEPHEN B. JOHNSON

Department of Medical Informatics, Columbia-Presbyterian Medical Center, New York, New York 10032

Received September 25, 1995

The issues and implementation of a clinical event monitor are described. An event monitor generates messages for providers, patients, and organizations based on clinical events and patient data. For example, an order for a medication might trigger the generation of a warning about a drug interaction. A model based on the active database literature has as its main components an event (which triggers a rule to fire), a condition (which tests whether an action ought to be performed), and an action (often the generation of a message). The details of implementing such a monitor are described, using as an example the Columbia-Presbyterian Medical Center clinical event monitor, which is based on the Arden Syntax for Medical Logic Modules.

INTRODUCTION

Clinical event monitors improve the quality and reduce the cost of health care by delivering information to health care providers when and where they need it. This delivery of information may take several forms. The monitor may warn a provider about a medication because of allergy, drug interaction, or side effect (*1–3*). It may interpret laboratory tests (*2*, *3*). It may remind a provider about following up on a screening test (*3*, *4*). It may suggest a diagnosis or new treatment option (*2*, *3*, *5*, *6*). The monitor may coordinate a complex clinical protocol, making sure that each provider is aware of his or her part in the plan (*2*, *7*, *8*). The goal of a clinical event monitor is not to replace the health care provider, but to make his or her job easier.

These automated guardians have been in real use for over 20 years (*9*), but until recently, their penetration into the health care market was minor. Now with more widespread computerization of health care institutions, stronger focus on health care cost and quality, and larger networks of health care providers, the interest in clinical event monitors has grown (*1–38*).

Using a clinical event monitor effectively can be a challenge, however. Each group that installs one ends up wrestling with similar issues. The purpose of this

0010-4809/96 $18.00

paper is to explore the theoretical and practical issues related to implementing and using a clinical event monitor, so that future users can benefit from past experience. The focus of this paper is not on the medical issues, but rather on the technical issues that arise from the medical requirements (for example, robustness, performance, flexibility). Where appropriate, examples will be drawn from the Columbia-Presbyterian Medical Center (CPMC) clinical event monitor (*10*) and from several clinical event monitors in the literature.

Requirements

A great number of problems in health care can be addressed according to a fairly simple scenario. In his or her course through the health care system, a patient engages in a series of clinically relevant events. Telephoning for an appointment, showing up at a clinic, being registered, being interviewed by a health care provider, being examined, having tests scheduled, undergoing tests, getting test results, having treatment prescribed, undergoing the treatment, and following up on progress are all examples of clinical events. The role of the clinical event monitor is to track the numerous events; pick out those about which it has some knowledge; look for clinically important situations; and, if appropriate, send a message to the providers. For example, the act of prescribing penicillin is an event. The event monitor may have in its knowledge base a rule that produces a warning when a patient is prescribed a medication to which he or she is allergic. If a penicillin allergy is discovered, a warning will be sent to the provider who wrote the prescription. The earliest and the most successful clinical event monitors have addressed this simple scenario.

Event Requirements

The phrase ''clinical event'' is very general, because it is intended to cover a broad range of occurrences. Anything that a patient does, and anything that happens to a patient can be considered an event. Most health care computer systems capture only a small fraction of occurrences that could be considered clinical events. The events that are most commonly captured today are registration events such as visit, admit, discharge, and transfer information; the receipt and results of laboratory tests; the distribution of medications by pharmacy; and the scheduling of major procedures. To a lesser degree, orders for tests and medications; actual dispensing of medications to patients; clinical observations of the patient (especially vital signs); and history, assessment, and plan for the patient may be available.

Clinical events can be defined even more broadly. The arrival of a provider to a location and the act of signing on to a clinical information system can be considered events. The occurrence of instants in time are events. They may be single events, such as the instant at 14:35 on October 10, 1995, or they may be a series of events, such as the arrival of winter or the first day of the month. Any of these events may trigger the clinical event monitor into action.

Data Requirements

The event monitor requires more than just the triggering event to come to most conclusions. For example, to send a message about a drug allergy, the monitor requires both a medication ordering event (for example, an order for penicillin) and access to patient data (for example, the presence of a penicillin allergy noted in an earlier encounter). Making a diagnosis, suggesting a treatment, and critiquing an action generally require a great deal of data.

As with events, only a small fraction of the clinical data that are really generated are captured in a patient database. Registration data, laboratory data, and pharmacy data are commonly available. Textual reports of discharge summaries, radiology exams, pathology studies, etc., are often available, but have limited use to a clinical event monitor without proper coding. Coded history, physical exam, assessment, and plan are available much less often.

A clinical event monitor will be only useful to the degree that the clinical encounter is captured, both as clinical events and as data in the patient database. This single issue, the difficulty capturing the clinical encounter, is the single most important stumbling block to the widespread use of clinical event monitors. The solution is beyond clinical event monitors themselves; it requires a concerted, expensive effort to collect clinical data in electronic form. As existing event monitors continue to show effectiveness, however, the incentive to collect data will increase.

Most health care organizations do not have a single, consistent, centralized patient database. Instead, data are available in bits and pieces, spread throughout the organization in various formats and coding schemes. Yet these data are critical to the clinical event monitor, so they must be made available either through interfaces or collection in a common repository.

Condition (Logic) Requirements

Once the monitor has been triggered by an event, it must carry out some sort of reasoning process to decide whether an action is warranted. In many cases, the reasoning is specified as a set of clinical criteria, which might include the presence of a disease, a laboratory test that exceeds a threshold, age limits, etc. If the criteria are met then a message is sent to the appropriate provider.

More complex diagnostic and therapeutic recommendations may require knowledge in a form that is difficult to express as a set of criteria. Such knowledge must often be elicited from experts and encoded in a knowledge base that support complex reasoning.

Protocols, including practice guidelines and care plans, deserve special mention. While they are often expressed as a set of criteria and actions, they are often difficult to implement due to very complex logic flow, sophisticated temporal reasoning requirements, and interaction among several health care providers.

An additional critical function of the event monitor's logic is to filter the incoming data. It is easy to forget that the raw data that are stored in a patient database require a great deal of processing to produce useful information. Raw

data that providers seem to use effortlessly must undergo explicit processing when used by an automated system. A simple example is the result of a blood test, such as the serum potassium. Without being conscious of it, a provider makes a number of checks to ensure reliable data: is the value recent enough to be relevant or is it a year old; is it a valid number or is the result ''hemolyzed''; is the number within a reasonable range or is it an obvious laboratory error such as 500 mEq/liter; and is it consistent with other values or has it undergone an unlikely change, such as several mEq/liter in minutes in a patient who is in no distress.

Action Requirements

Once the criteria are met, a number of different actions might be indicated. A message may be sent to a provider by electronic mail, through a paging system, through the electronic patient record, via a facsimile machine, over surface mail, or through a human intermediary. The method will depend on the urgency of the message and the availability of the provider. If a provider does not acknowledge that a message was received, then an alternate route may be chosen. A message may instead go to a patient or the patient's family. It must be possible to review all messages that have been sent for a patient.

Actions besides the generation of messages are useful. The monitor may interact with other computer systems. For example, the event monitor may help determine when the paper chart should be scheduled for retrieval before a clinic appointment. It might help allocate resources in a clinic, such as scheduling the necessary supplies for the procedures that are expected the next day.

Despite its name, a clinical event monitor is often used for administrative reports and functions: monitoring length of stay, measuring resource usage, coding records, checking insurance eligibility, etc. Event monitors can play a major role in quality assurance, such as detecting adverse drug events.

Performance Requirements

Clinical event monitors derive their benefit from serving as tireless observers, constantly monitoring clinical events. Without adequate performance, the monitor will fall behind, resulting in tardy messages. Once this is recognized, implementors will tend to limit the purposes the system is used for, thus limiting its benefit. The monitor must perform well enough to handle the expected workload. Then it must have sufficient reserve to handle the new uses that providers think of.

This is no easy task, however. At a large organization like Columbia-Presbyterian Medical Center (50,000 admissions and 700,000 outpatient visits per year), for example, 7000 laboratory tests are performed per day. Each test is associated with several events: ordering, collection, acknowledgment of receipt, execution, preliminary reporting, and final reporting. Pharmacy has a similar magnitude of events, and all the other ancillary departments combined represent another similarly sized group. If it were possible to capture history and physical informa-

tion, there would be an additional huge load. We estimate the need to handle 10 events per second at peak times at CPMC.

Reliability Requirements

Given the number of events processed per day, it is possible for a small error to cause the event monitor to generate a large number of erroneous messages. Such a problem occurred once at CPMC, when it generated 93 erroneous tuberculosis warnings in a few minutes (*39*). Such errors undermine confidence in the system. Errors in which messages fail to appear or contain erroneous text are also damaging. The monitor must be reliable.

Because the monitor interfaces with so many systems—applications that produce events, databases, and routing applications—the potential for error is great. Besides the usual concerns about computer system failures and network failures, the difficulty coordinating the many different groups responsible for the systems can lead to further problems. For example, a change in the laboratory coding scheme will lead to errors in the event monitor if it is not modified to accommodate the change. Even if tools and procedures are available to reduce these errors, some will always occur, and the event monitor must have a means to handle them gracefully.

Flexibility and Ease of Use Requirements

The medical knowledge represented in the clinical event monitor is complex and the environment that the monitor sits in is complex, including all the systems the monitor interfaces to. Changes are constantly occurring in medical protocols, laboratory instrumentation, disease patterns, institutional priorities, ancillary departmental computer systems, communication protocols, etc. The event monitor must be capable of accommodating many different types of knowledge, events, and data, and it must be easy to maintain.

The acquisition of clinical knowledge from experts and databases must be made as easy as possible. The acquisition of knowledge is an enormous bottleneck to the expansion of the monitor to its potential.

Model

While the earliest uses of event monitors were in health care (*9*), in more recent years event monitors have been put to more general use. The following model is derived from computer science's active database literature (*40–42*).

An active database consists of a standard database (usually relational) and an event monitor. The event monitor, in turn, consists of a set of rules and mechanisms to run the rules against the database. Events such as database operations cause the triggering of a rule. The rule's condition consists of queries to the database. If the condition is satisfied (at least one query is non-null), then the rule's action, which consists of other database operations, is executed. Active databases have been used, for example, to maintain database referential integrity.

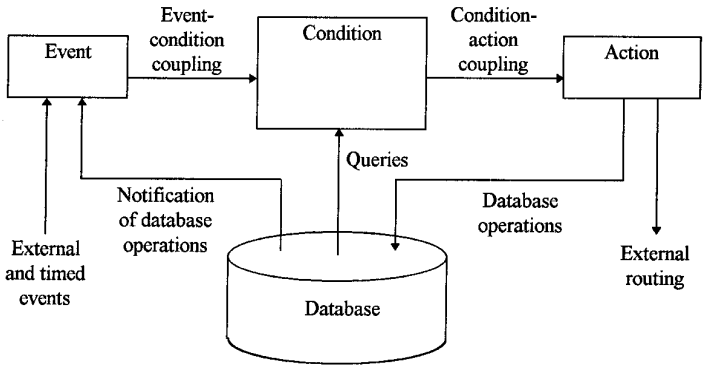

FIG. 1. Clinical event monitor overview. Database operations and external events trigger the evaluation of a condition. Based on the event and additional patient data, the condition determines whether an action ought to be performed.

We have had to make only minor modifications and extensions to the basic active database model to better accommodate health care requirements. In this extended model, a rule consists of the following components:

- rule identifier—unique handle for the rule
- mapping (data) definition—defines the mapping from the rule components to the database and external environment
- event (evoke) definition—what triggers the rule
- event-condition coupling mode—event and condition transaction linkage
- condition (logic) definition—criteria that determine whether to perform the action
- condition-action coupling mode—condition and action transaction linkage
- action definition—operation to carry out if the condition is met
- timing constraints—deadlines, priorities, urgencies
- contingency plans—in case of error or if time constraints are not met
- attributes—supplies further context for the rule

The flow is shown schematically in Fig. 1. The components are defined in further detail below.

The research in active databases has been focused on performance issues such as transaction modeling, efficient evaluation of overlapping conditions, scheduling transactions, active DBMS architecture, nested transactions, and performance monitoring. The focus of research in clinical event monitors has been different. The knowledge base is important because of the vast number of potential rules and the fuzziness and complexity of those rules. The database serves a critical role because of the amount of data and the different types of data. Performance has focused largely on database access.

RULE KNOWLEDGE BASE

How to represent rules is a long-debated issue. It is possible to code the rules in the programming language of the clinical information system (*1*). This is useful

```
maintenance:
      title: Alert on low hematocrit;;
      filename: low_hematocrit;;
      version: 1.00;;
      institution: CPMC;;
      author: George Hripcsak, M.D. (hripcsa@cucis.columbia.edu);;
      specialist: ;;
      date: 1993-10-31;;
      validation: testing;;
library:
      purpose: Warn provider of new or worsening anemia.;;
      explanation: Whenever a blood count result is obtained, the
            hematocrit is checked to see whether it is below 30 or
            at least 5 points below the previous value.;;
      keywords: anemia; hematocrit;;
knowledge:
      type: data-driven;;
      data:
            blood_count_storage := event {'complete blood count'};
            hematocrit := read last {'hematocrit'};
            previous_hct := read last ({'hematocrit'}
                where it occurred before the time of hematocrit);;
      evoke: blood_count_storage;;
      logic:
            /* check that the hematocrit is a valid number */
            if hematocrit is not number then
                  conclude false;
            endif;
            if hematocrit <= previous_hct-5 or hematocrit<30 then
                  conclude true;
            endif;;
      action:
            write "The patient's hematocrit ("|| hematocrit ||")
                  is low or falling rapidly.";;
end:
```

FIG. 2. A sample Arden Syntax Medical Logic Module

to get a quick start and prove the concept, but the relative difficulty coding new rules leads to other alternatives.

Several successful clinical event monitors have used procedural rules languages specifically designed for health care. HELP (*43*) and CARE (*3*) are two early ones. The Arden Syntax for Medical Logic Modules (*11*) grew out of the two earlier languages. The Arden Syntax is also based in part on the active database literature; this is reflected in its slots, which closely match the components of an active database rule: filename, evoke, data, logic, action, etc. Arden Syntax rules are called Medical Logic Modules. An example is shown in Fig. 2. A detailed

discussion of the Arden Syntax and its use in event monitors has been published (*44*, *45*).

An extension to the SQL relational database query language has also been used in active relational databases. This has the advantage that it is a standard (except for its extensions), but coding health rules in SQL can be tedious.

Nonprocedural rule languages have been used (*22*, *29*, *32*, *34*, *37*). One such system (*29*) that is in wider use boasts the ability to use fuzzy logic, but its inability to handle lists of information (for example, the last 10 potassium levels) and its limited ability to manipulate data (for example, convert raw codes to a usable form) make it less useful in health care.

Which representation is best depends on the task at hand. The procedural languages are easy to implement, easy to learn, and powerful enough to convert the raw data found in databases to useful information. For discrete, modular rules with many links outside the event monitor itself, a procedural language is a good choice. For knowledge bases with a huge number of rules composed of trivial queries, a language based on SQL or simple declarative clauses is appropriate. For systems with complex protocols, a language specialized to the purpose may be appropriate.

The discussion about knowledge base representations will continue and is worthy of far more space. The reader is referred to other discussions (*22*, *23*, *26*, *44*, *46*). The focus of this paper is on the general issues of implementing a clinical event monitor. No matter which representation is chosen, many of the implementation issues are similar because much of the difficulty lies in the interface to the outside world (events, database queries, message routing), and this interface is necessary regardless of the rule representation. At CPMC we use the Arden Syntax, and the discussion below on rule condition will focus on procedural languages.

Knowledge Base Maintenance

The following basic operations are allowed on rules:

- create—create a new rule
- update—modify a rule
- activate—enable a rule to be triggered
- deactivate—disable a rule from being triggered
- fire—performed by the system when a triggering event occurs

While rules can be deactivated, and therefore prevented from firing, rules generally cannot be deleted. This allows a record to be kept of all rules that have ever fired.

Event Component

Event Types

A clinical event monitor must support a range of event types. A summary of event types follows:

- database operation—insert, update, delete, select, etc. (before or after the operation)
- temporal event—absolute, relative, periodic
- abstract event—detected and signaled by users or external programs
- composite event—sequences of other events

The storage of data in the patient database represents a common and important source of events. In health care, rules are usually triggered after the database operation. For example, after a laboratory result is stored, a rule might check whether the result is within acceptable limits. Less commonly, a rule may be triggered before the database operation, usually to set up some context for the database operation itself.

Temporal events include absolute points in time, points in time that are defined relative to some baseline event (for example, 3 days after surgery), and periodic series of points in time. Abstract events refer to all events that originate from outside the system. The most common sources for such events are clinical applications. Composite events consist of sequences of other events.

Event Notification Message Definition

The event notification message serves several purposes. Its primary purpose is to tell the event monitor what type of event just occurred, so that the appropriate rules can be evoked. The message can pass pertinent event information to the rule, and it can pass pertinent application information to the rule.

The following event notification message format is used at CPMC:

Event definition

- medical record number—defines the patient
- event type—order, result, ADT, diagnosis, etc.
- event code—blood count, admit, pneumonia, . . .
- event time—time that the triggering event occurred

Event data—for events involving the storage of data

- event data accession number—unique key for the data in the database
- event data primary time—for example, time that a sample was drawn
- event data status—for example, preliminary, final
- event data mode—specifies whether the data are available in a cache

Application information—refers to the application that generates the event

- evocation mode—defines application-specific event-condition coupling
- routing mode—allows the application to override the rule's default routing
- routing destination—allows an application to receive a message
- now—the monitor may run as of some time in the past; used for debugging

Triggering information

- trigger mode—normal event driven or a direct call to a particular rule
- rule identifier—name of the rule to be called in direct mode
- evocation count—prevents runaway rules

Events are defined by a pair of codes: the event type and event code. The distinction was made to avoid redundancy in the event definitions. Many clinical

events represent an action or operation between the patient and a clinical entity. The event type defines the action and the event code defines the entity. When a chest X-ray report is stored in the database, the event type is clinical data storage and the event code is chest X-ray. Because one clinical entity may participate in many different actions (event types), this separation reduces redundancy. For example, a blood test may be ordered, acknowledged, collected, received, processed, stored as preliminary result, and stored as final result. For some clinical events, such as temporal events, there is no reasonable event code, just an event type.

The patient to whom the event occurred is defined by the medical record number. The event time tells when the event occurred, such as the placement of an order or the posting of a result.

Some events, such as the storage of a laboratory result, are associated with patient data. In this case, the event notification message contains additional information about the event and its associated data. The accession number tells the rule exactly which data in the database were associated with the current event. The primary time is the most medically relevant time for the data associated with an event, and it is generally different from the event time. For example, the posting of a laboratory result has as its event time, the time that the data were posted but as its primary time, the time that the sample was drawn from the patient. (The primary time and status are included for increased performance; the accession number is sufficient to define the data uniquely.)

The application information allows an application to override the way a rule is executed. For example, an order entry application may cause a rule to send its action's messages back to the application (so it can display the messages to the user) instead of the database. The triggering information allows a rule to be called directly. It is often used for debugging.

Database Event Generation

Implementation depends mainly on whether the patient database has database triggers available. The Sybase database management system (*47*) was among the first commercial databases to support database triggers, but many others are adding the feature. Given such a database, there are three options: use the system's triggers and rules; exploit the triggers but use external rules; or do not exploit the feature at all. The first option is the quickest and easiest to implement. Database triggers are captured by the system at a very low level, improving efficiency and permitting application developers to write data manipulation queries directly against the database. The disadvantage is that the rules are generally limited in their ability to carry out complex inferencing and to interface to the outside world.

The second option relies on the database management system to notify the event monitor that a database event has occurred, but carries out rule processing outside the database. The advantage is that triggers are still captured at a low level, but the inferencing can be as complex as desired. That option is only available if the triggers are flexible enough to trigger external applications.

The third option, which is the only one available for database management systems without triggers, implements the entire event monitor outside the database. All applications that manipulate the database must notify the event monitor of the updates. This is less efficient than an internal mechanism and is prone to error when developers fail to perform the notification. In some clinical information systems, application developers are forced to make all database updates through a small number of data access routines. These routines automatically notify the event monitor of the updates, so the risk of missing events is reduced.

Temporal Events

Temporal events can be implemented with a temporal trigger database and a system clock. The database need only have a trigger time and the appropriate event notification message. Whenever the system clock time exceeds the trigger time, a daemon causes the event monitor to process the event notification message. How often the daemon polls the temporal trigger database depends on how accurate the actual triggering time must be. In most clinical applications, accuracy to a few minutes is sufficient, because they are not synchronous with a triggering application anyway.

While it may be possible to implement temporal triggers with operating system primitives, a separate temporal trigger database is preferred. Operating system primitives may not be robust against system crashes, and they are often limited in how long a time period they can accept.

Absolute time triggers are straightforward. The desired trigger time is simply inserted into the temporal trigger database. Those event monitors that execute rules in batches at night (*3*) can be represented as having a nightly event that trigger all the rules (although one would not implement it this way).

Relative time triggers are defined with respect to an initial baseline event. The baseline event, which can be of any type, triggers the system to add the desired time delay and place a record in the temporal trigger database. A separate program can perform this task, or, if rules are powerful enough, a rule can carry out the operation.

Periodic time triggers can be implemented similarly to relative time triggers. One method is to write the repeated triggers to the temporal trigger database all at once. Another method is to write records one at a time; as each record is executed, the next one is written. The advantage of the latter approach is that it is easier to terminate the series of triggers early. For example, if a rule is defined to fire every day for 10 days *until* a discharge summary appears, then if the summary appears on the first day, the nine remaining triggers in the database need not be deleted (or ignored).

Abstract and Composite Events

To create an abstract event, an application merely needs to send an appropriate event notification message to the event monitor. If the event monitor has been implemented entirely out of a database management system's trigger and rule

mechanism, then it may be necessary to write a dummy message to the database to create a trigger.

The definition and implementation of composite events is complex. Generally, the goal is to trigger when a particular sequence of events occurs. The rule author must be able to specify an event ordering and time constraints on the maximum time acceptable between events. For example, one might trigger a rule if a patient undergoes a chest X-ray within 1 day after admission to a hospital. The pair of events, admission and chest X-ray, would be specified with a 1 day maximum time interval. While this is a convenient way to specify this example, a general implementation of all possible scenarios may not be worth the effort. It requires tracking the state of many partially completed events (for example, all admissions would need to be tracked pending the arrival of a chest X-ray).

An easier way to implement composite events is to support only simple events through the triggering mechanism and let the rule inferencing mechanism simulate complex events. For example, a rule that is triggered by chest X-rays could check to see if the patient had been admitted within the past day.

Event–Condition Coupling

Modes

Event–condition coupling defines how the transaction that generated the event (for example, storage of a potassium result) is linked to the transaction that evaluates the rule's condition (the event monitor transaction). If they are part of the same logical unit of work, then an error in the rule evaluation will roll back the triggering event (undo the potassium storage). This is often undesirable in health care. In most real systems, if the event monitor is not working, one would not want to halt all storage to the patient database. The extreme importance of allowing health care providers to review clinical data usually overrides any concern about missed event monitor messages. The lost messages can usually be recovered by resending the data when the event monitor is again functioning.

At CPMC, the storage of data and event processing are always separate logical units of work. Coupling modes are still useful, however, for defining when and how the event processing occurs:

Asynchronous. In this mode, the event and the rule evaluation are done in separate, independent transactions. The application that generates the event merely gets back an acknowledgment that the event notification message was posted successfully. Systems that generate clinical data, such as the clinical laboratory, call the event monitor in asynchronous mode. Since there is no user to show the generated alerts, there is no need to wait for the event monitor messages.

Synchronous. In this mode, rules are evaluated within the same transaction as the event, but in a separate logical unit-of-work (that is, a commit statement lies between the event and the rule evaluation). The effect is that the application that generates the event waits for the triggered rules to be evaluated, but if rule evaluation fails, the event is not rolled back. This is used by applications that want to show its users the messages generated by the event monitor. Even if the

event monitor fails, the event will still be committed (for example, data will still be written to the patient database). When such an error does occur, the application may warn the user that the event monitor is not functioning properly. The above discussion concerns only those rules that are triggered immediately by an event; rules that are triggered some time delay after an event must, by definition, remain asynchronous. Messages generated after a delay will not be sent back to the calling application. The synchronous mode is used by applications in which the user enters data such as physical exam observations. When the data are stored, the user is shown any pertinent event monitor-generated message, but if the event monitor is not working, the data are still stored.

Hypothetical. This mode is similar to the synchronous mode, but it prevents any rules from running asynchronously. Rules that would have been evoked after a delay are not triggered. By also altering the rule's routing of messages (see below) so none are actually sent to providers (part of the routing function), this mode allows an application to ask a hypothetical question: what would happen if this event were to occur? This is used to check for alert messages before processing an action. For example, before an order for a medication is stored, an application can ask the event monitor whether any alerts will be generated because of the order. If so, they are shown to the ordering provider, who then has a chance to cancel the order. Because no messages are actually routed and because no delayed rules will generate messages later on, the canceled order will have no lasting effect on the patient record. If the provider confirms the order, then a new event is generated which is no longer hypothetical.

In the hypothetical mode, it appears to the clinical user that there is a logical link between the triggering event and the event monitor messages. The application can be written so that a user cannot commit an event (a medication order) unless he or she acknowledges the event monitor's messages (medication warnings). This link, however, is accomplished at the application level rather than using a single logical unit of work.

Rule Triggering

The event monitor must trigger the rules that are pertinent to the triggering event. The mechanism depends in part on how triggers are implemented, but most systems look similar (Fig. 3). When a rule is activated (that is, turned on so that it can be triggered), its event definition is extracted and stored in a trigger database. When event notification messages arrive, a query to the trigger database reveals which rules are pertinent to the event. The rules are executed, usually in arbitrary order. To gain efficiency, the trigger database may be read to a local cache.

Mapping (Data) Component

Mapping

While active databases do not have a "mapping" component, it is useful for clinical event monitors to distinguish the mapping component from the others.

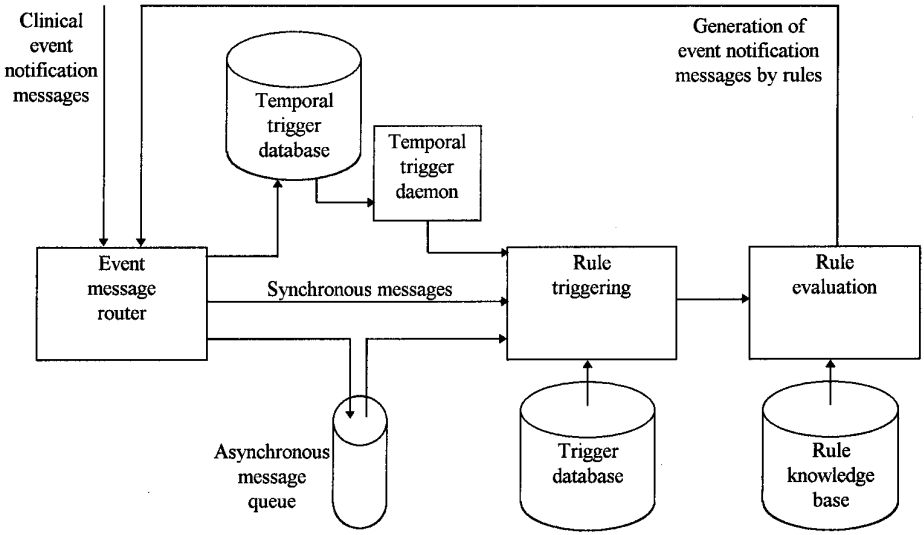

FIG. 3. Rule triggering mechanisms. Events may be linked asynchronously, synchronously (including hypothetical), or through a time delay to the rule triggering mechanism. The trigger database determines which rules are pertinent to the event.

Active databases are generally self-contained systems that can be expressed in a single (usually relational) model. A major focus of clinical event monitors is to interact with the outside world. The clinical databases are often external to the event monitor itself, and there may be several database models in use at once. The added complexity of an event monitor's environment justifies the distinction.

The mapping component contains the mapping from objects referred to in the event, condition, and action components to entities in the database and in external applications. The most common mappings define queries to the patient database, but other mappings include event definitions, routing specifications, and links to other rules.

A number of major issues concern database queries. They are largely outside the scope of this paper, but will be mentioned here in brief. Vocabulary is a difficult and challenging issue. The variety of data that can be stored in a patient database is enormous. The number of terms necessary to code all the data probably exceeds 100,000. Furthermore, the terms have complex relations with each other. Methods and tools are available to manage such unwieldy vocabularies (*48–50*). Terms are usually organized into a class hierarchy. By exploiting classes, a rule can reduce redundancy and maintenance. Knowing the vocabulary term is necessary, but not sufficient. One must also know where and how it is stored in the database. The database schema holds this information. The vocabulary and schema will vary from institution to institution.

Aside from defining what one wants via the vocabulary and schema, there are a number of constraints and operators that we have found generally useful for

querying clinical data at CPMC. To understand a CPMC query, one must first understand the basic schema (*51*). Clinical objects are split into header and components. The header contains information that is common to all the components, such as the object type code, patient identifier, relevant times, status, and unique database key for that object. The components contain a component code, a value, and a link to its header; components may be nested. For example, a blood chemistry battery has a code in the header that identifies it as a chemistry battery. The individual tests are stored as components; so there would be a code for blood potassium and a value. Each clinical area (laboratory, pharmacy, patient management, textual reports, etc.) has its own header definition, because different attributes are relevant for different areas. Given this schema, a standard clinical query has the following fields:

Patient identifier
- medical record number—uniquely identifies a patient

Data definition
- data access module name—specifies the clinical area
- header codes—selects desired header objects
- component codes—selects desired components

Query by key
- key—for retrieving particular results, if one has its key
- type of key—database key or ancillary department accession number
- evoking—used when a data storage event triggers a rule; this parameter requests that the data that caused the triggering be retrieved

Time constraints
- begin time—beginning of search interval
- end time—end of search interval
- begin operator—specifies inclusive or exclusive for beginning of interval
- end operator—specifies inclusive or exclusive for end of interval

Other constraints
- status value—constraint data to particular statuses (final, preliminary, etc.)
- constraints—five single-character parameters specific to the clinical area (for example, in- or outpatient for patient management data)
- auxiliary string—forty characters for passing additional constraints for a clinical area

Projection operators
- display header—return attributes stored in the header (primary time, update time, header code, database key, accession number, status, alert flag; for pharmacy: start time, stop time, frequency, quantity)
- display component—return attributes stored in the components (component code, sequential component number, parent pointer, value type, value)

Aggregation operator
- aggregation function—chooses a function (none, last, first, min, max)
- aggregation number—how many results should be returned

Sorting operator
- ordered by—what to sort the results by (primary time, update time, status)

The event component of a rule specifies the rule's triggering. It may be more convenient **not** to refer to institution-specific event codes in the event component, but rather to use the mapping component to map general terms (''storage of potassium'') to institution-specific event codes. The general terms can then be referred to in the event component. Placing the event mappings in the mapping component has two advantages. Because the condition and action can also refer to clinical events (for example, the condition may ask whether a particular event ever occurred on a patient, and the action may signal an event), grouping all the event mappings in the mapping component simplifies maintenance and may reduce redundant definitions of clinical events. As explained above, at CPMC events are defined by a pair of codes: an event type and an event code.

So many rules are triggered by the storage of data that in many systems the distinction between events and data is blurred. For example, one may retrieve the blood hematocrit and be triggered by the blood hematocrit. In fact, the rule is actually triggered by the storage of a blood hematocrit result. Two rules that refer to the same data (hematocrit) may be triggered by different events (storage of the result versus acknowledgment that the specimen was collected). It is therefore better to distinguish events from data.

The disadvantage of the distinction is that many mappings appear redundant: the definition of hematocrit and the definition of a hematocrit storage event. While we do not use the following method at CPMC, one can use implicit event definition to reduce the redundancy. Whenever the result of a data query is used in a context where an event was expected (such as the rule's event component), the event monitor assumes the rule is referring to a data storage event.

The action component refers to entities in the external environment to perform tasks like routing. One may also use the mapping component to map general terms to actual entities in the external environment (for example, a program name), or one may perform these mappings in the action component itself.

Query Optimization

In order to achieve effective optimization, the primary bottlenecks must first be determined. A clinical event monitor is triggered by discrete events. We will look at the time to process a single event. One event may trigger many rules to execute. Each rule may require any number of independent data elements, usually belonging to the patient to whom the event applies. The time to process an event (t_e) depends on the time to perform initialization tasks (t_i), such as determining which rules are to be triggered, the number of rules that are triggered (n), the time to process a rule (t_p) not including queries or routing, the average number of queries performed per rule (q), the time to perform a query (t_q), the average number of messages generated per rule (m), and the time to route a message (t_r).

$$t_e = t_i + n(t_p + qt_q + mt_r)$$

In most systems, the initialization time is insignificant compared to the time

to process the rules and to perform queries. Systems designed to generate alerts, interpretations, and reminders tend to use simple logic that requires little processing time. Since only a small number of rule triggerings tend to result in message, the time to route messages is also unimportant. In these systems, the time to perform queries is the critical bottleneck, and they are called ''query-bound'' systems.

In systems with more complex reasoning, especially those with recursive or iterative algorithms, the processing time is the greatest factor (*30*), and these are called ''processing-bound'' systems. Systems that keep the entire active patient database in a local cache are also likely to be processing-bound.

To improve performance on query-bound systems, one must reduce the time to perform queries or reduce the number of queries. There are several approaches. The time to perform queries can be reduced by locating the clinical event monitor close to the patient database, thus eliminating network delays and simplifying access.

By caching data, one can reduce the time to perform queries dramatically. One can read the entire record for a patient or a relevant portion of it (based on the context set by the triggering event) into a cache before the rules are processed. The bottleneck then becomes reading this chunk of the record. One can include the time to read the record in the initialization time (t_i). If the time to read the record is less than the time to perform all the queries for all the rules (nqt_q), then this method is worthwhile. In some systems, especially those operating over a network, the overhead of querying across the network (e.g., initializing and terminating the connection) is greater than the time to perform the query on the server or to transfer the bytes over the connection. In these cases, transferring the patient record as a chunk is faster.

Another approach comes from the observation that the triggering event sets the clinical context for the rules, so one would expect the triggered rules to retrieve similar data elements. Any data that are retrieved can be cached so that other rules may benefit. A special case of this can be carried out for events that represent the recording of clinical data. For the rules triggered by such an event, the most common query is to retrieve the data that were stored as part of the event. By simply passing the data from the event manager to the rules in a cache, a large portion of queries can be reduced to insignificant time.

The number of queries can also be reduced. What matters is not the number of queries defined in a rule, but the number actually performed. An effort should be made to come to a decision on whether or not to send a message based on as few queries as possible. By reordering logic and deferring query execution, the time can be reduced. Fast queries should be executed first. For example, if one of a rule's queries refers to data already stored in a cache because of an earlier rule, then the rule should try to come to a decision based on those data, rather than performing another query to the database. Because of the dynamic nature of the cache, query deferral is best accomplished at run time. It is often easier to implement, however, at compile time, and if the optimization algorithm is complex, it may be too slow to carry out at run time.

The CPMC clinical event monitor is a query-bound system. A study (*52*) revealed that a rule had a median query time of 36 msec but a median CPU time of only 2 msec. We choose to locate the event monitor on the same computer as the patient database (IBM 3090 mainframe computer). We use compile-time query reordering.

Condition (Logic) Component

Making Decisions

The condition component of a rule serves three major purposes: making decisions, filtering, and variable binding. Its basic function is to decide, based upon the triggering event and patient data, whether to perform an action. As described in the ''Rule Knowledge Base'' section, the method of inferencing depends on the event monitor. The reader is referred to other discussions on the issue of knowledge representation (*22*, *23*, *26*, *44*, *46*). The CPMC event monitor uses the procedural logic of the Arden Syntax.

Assuming procedural logic or simple declarative conditionals, the following basic features are necessary at the very least. There must be a way to refer to data defined in the data component of the rule (in the case of active databases, there will be direct references to the database). There must be basic data manipulation operations, such as simple arithmetic to process the data; comparison operations to test for existence, equality, and thresholds; and Boolean operations to combine criteria. Finally, there must be some way to assert whether the action should be taken. This may be based on the value of the conditional expression, or it may be stated explicitly, as with the Arden Syntax conclude statement.

While the above represents the minimum necessary for a useful system, the data manipulation operators, comparison operators, and logical inferencing may be expanded. At a certain point, however, the added complexity of the language detracts from usability; the right balance must be struck.

Explicit handling of time is an important addition. The Arden Syntax supports temporal operators over points in time and implicit intervals (*44*); other systems have addressed abstract and fuzzy times (*53–55*). A few systems have added fuzzy logic support to the inferencing mechanism (*29*). Explicit communication among rules, for example for scheduling protocols (*7*), may be supported; the Arden Syntax uses direct calls and event signaling for this purpose.

Filtering

The second purpose of the condition component is to filter raw data to make them useful to the inferencing mechanism. To serve as an effective filter, the event monitor must have several capabilities. It must have a considerable ability to handle multiple data types, data type conversion, and data manipulation. At CPMC, we have found that in real patient databases, multiple representations of the same abstract data type always seem to creep in. A simple example is time. We use a relational database that supports a timestamp data type, but we

find in various fields supplied by clinical applications other time formats including strings and numbers. This occurs because developers of the event monitor are rarely in complete control of all institutional applications. For example, an application may store age as a string (''3 months'') rather than the birthdate. The approximate birthdate must be calculated from the age and time of the observation (which does use the database's timestamp data type). The ability to manipulate strings is critical in such areas.

The event monitor must be able to aggregate lists of data. For example, a series of observations may be put through a trend analysis to decide whether an abstract condition is present; a series of creatinines may help decide the presence of worsening renal insufficiency. The simple aggregation available in SQL, such as count, min, max, sum, and average, is not sufficient. There must be a convenient way to eliminate outliers, perform statistical analyses, reorder values, etc.

Procedural constructs such as the ability to loop may be valuable, but certainly add to the complexity of rules. They may be handy for sophisticated filtering and for carrying out complex actions, but they only make sense in a procedural representation.

At CPMC, the filtering logic is contained in the rules, but it would be possible to create a subsystem whose sole purpose is to perform filtering (*31*). The rules would then be able to refer to abstract concepts supplied by the subsystem. This would simplify the rules and reduce filter redundancy.

Binding

The third purpose of the condition component is to bind values to terms that may be used in the action component. In this way, messages may be tailored to the patient. This is usually done by assigning values to variables. The condition often refers to the same values in the process of making a decision, so the ability to assign values as part of the condition and reuse them in the action reduces redundancy. It also tends to simplify the action component.

Implementation

The implementation is very dependent on the knowledge representation. At CPMC, we have implemented our procedural logic with pseudo-codes (*10*); the same approach has been taken by others (*56*). The knowledge base is compiled into an intermediate representation (pseudo-codes) that can be executed on a virtual (software-based) machine. This method has the advantage of quick implementation and easy transportability across platforms. To add a new platform, one need only migrate the virtual machine, which is generally a straightforward program. The compiler and knowledge engineering tools may be shared across platforms. (The advantage of portability has become less important with the emergence of C++ across platforms.) Because the pseudo-codes are interpreted, however, execution of the logic may be slower. We have found that database queries are such a bottleneck that the time spent executing pseudo-codes is negligible (*52*).

TABLE 1

PSEUDO-CODES USED IN THE CPMC EVENT MONITOR VIRTUAL MACHINE

Exit	Arccos	And	Catenate
Jump	Arcsin	Equal	First
Jump if	Arctan	Not equal	Last
Drop	Cos	Less than	Take
Pick	Sin	Less equal	Element
Dup	Tan	Greater than	Compress
Roll	Log	Greater equal	Upgrade
Swap	Log10	Append	Downgrade
Null	Exp	Plus reduce	Index
Literal boolean	Int	Minus reduce	Call mlm
Literal integer	Abs	Times reduce	Call event
Literal number	Sqrt	Divide reduce	Route
Literal code	Is null	Power reduce	Index of
Literal time	Is boolean	Or reduce	Qcall mlm
Literal month	Is number	And reduce	Qcall event
Literal second	Is code	Equal reduce	Tcall mlm
Literal string	Is time	Not equal reduce	Tcall event
Pfetch	Is duration	Less than reduce	Query
Fetch	Is string	Less equal reduce	Ravel
Pstore	Plus	Greater than reduce	Noop
Store	Minus	Greater equal reduce	Current time
Argument	Times	Append reduce	Mlm code
Return	Divide	List append	Med query
Not	Power	Is list	
Negate	Or	Shape	

The virtual machine used at CPMC is a stack-based machine. The pseudo-codes for the machine are shown in Table 1. The ''literal'' operators place literal data on the stack. The ''reduce'' operators perform binary operations sequentially on items in a list, so ''plus reduce'' finds the sum. The operator names are derived from the language Forth (a stack-based language) and APL (an array-based language appropriate for list-handling).

The Arden Syntax has also been implemented in several programming languages, including Prolog, Smalltalk, MUMPS, C++, C, and APL. C++ is the most common language for implementing the Arden Syntax today (*19*, *28*). This is due to the availability of C++ across platforms and the ease of implementing Arden data types in the C++ class hierarchy.

ACTION COMPONENT

Messages

Several types of actions are available to rules. The most common action is some form of message. Messages may convey medically relevant information to a health care provider, to the patient, or to an appropriate organization (for example, a department of health).

Messages may be stored in the patient record or routed to a particular individual. In most systems, messages that are routed to an individual are also stored in the patient database as a legal record. Given modern networks and devices, a number of routing methods are readily available. Electronic mail, alphanumeric pagers (dialed through a modem), facsimile transmission, and even computer-generated voice messages can be sent automatically by the event monitor. A printed report may be mailed or inserted into the paper chart. A message may be integrated into the clinical information system; a message may appear on a terminal at the patient's location (for example, an ICU terminal), it may appear to a user when he or she logs on to the system, or it may be attached to relevant patient data (for example, the event data that caused a rule to be triggered).

The choice of routing method will depend on the urgency of the message, the availability of the individual, and knowledge of who to send the message to. An urgent message might be sent by pager, whereas a passive informational message may simply be attached to the chart. The most difficult problem in many institutions is simply figuring out whom to send the message to (*1*). For example, the admitting physician of record is often not the most relevant physician later in the hospitalization. In many hospitals, it is difficult to figure out who is in charge even by reading the paper chart; they are a far cry from capturing the information electronically. A number of solutions have been suggested, including sending messages to whomever has recently looked at the patient's electronic record (*1*).

It may be useful to define a set of message types. These types may be used to determine the routing and urgency of a message. At CPMC, we use the following types:

- alert—an urgent message that should be routed by the quickest means available (for example, drug interaction)
- interpretation—a passive informational message that can be called up by the provider when needed (for example, anion gap calculation)
- maintenance—messages that are routed directly to a health maintenance information system (for example, mammography is overdue)
- screen—a message generally sent over electronic mail to researchers or the quality assurance office indicating that a patient has fulfilled some criteria
- state—a message that conveys state information to other rules; it is stored in the database but not sent to any user (for example, patient is in step two of a protocol)

The urgency may be further modified by explicitly setting an urgency level (a number from 1 to 99 in the Arden Syntax). The routing of a message will be affected by the rule's validation level. The following levels are defined:

- production—rule may send any message
- research—rule may send screen messages, but may not send message to the patient or the patient's health care provider
- testing—rule may not send messages other than for debugging

A given rule may always send the same message, or the message may be modified according to the context set up by the condition of the rule. This may

be accomplished by binding variables within the condition section and using those variables to assemble the message.

There are several options available for storing messages in the electronic patient record. They may be stored as readable text. This is the simplest method, since querying applications may read the message and display it to the user without further processing.

Coded data offers several advantages, however. Space may be saved by storing a single code rather than an entire text message. Errors in message phrasing can be fixed in a single message table rather than in many patient records. Other rules and applications can use the coded data for further decision support. When messages are assembled using data bound in the condition section, the database will have to contain the bound data as well as the message code. Care must be exercised when altering a rule. Other than to correct an error, if a rule's message changes, then it should be assigned a new code. Otherwise the messages will change inappropriately for patients who received messages under the original rule. At CPMC we began with text messages and now support both text and coded types.

Other Actions

A rule may cause other rules to execute in several ways. The storage or routing of a message, like any other clinical event, may trigger a rule. Strictly speaking, this is not an action itself, but merely a property of the system. A rule may trigger other rules (or itself) directly. This is often used by rules whose function is to schedule other rules, often as part of a large protocol. It gives the opportunity for the scheduling rule to pass parameters to the other rules. A rule may also signal an abstract event. In this case, the event monitor receives an event notification message, which in turn triggers relevant rules. This allows one rule to trigger a group of rules easily. A rule may start other applications or queue a message to a running application.

To reduce redundancy, a single rule may perform multiple actions. The rule may choose among several actions based upon data bound in the condition section. While this is useful, one must be careful not to create enormous rules that are better split up into separate pieces.

Condition–Action Coupling

By definition, the condition does not carry out any operations that could be rolled back (for example, a database update), so the link between the condition and action is not so important. When viewed together with the event-condition coupling, however, the two coupling options determine how events are coupled to actions (that is, events to conditions and conditions to actions). For example, the storage of clinical data (event) might be rolled back because an electronic mail message could not be sent (action).

We have found that the event-condition coupling options are generally sufficient to determine how events and actions are coupled. Condition–action cou-

pling is largely determined by the action itself. For example, an electronic mail message is necessarily asynchronous. Those actions that are naturally synchronous, such as storing a message in the patient database, are performed as part of the same logical unit of work as the condition.

The triggering of other rules within one rule's action does have condition–action coupling options. Such triggering can be synchronous with the original rule (part of the same transaction), asynchronous (part of a separate transaction), or delayed (part of a separate transaction and at a later time).

Error Handling

In any environment, there will be errors. It is obviously important to minimize bugs in the clinical event monitor and system down time. Even with no bugs in the event monitor itself and with a perfect operating system and hardware, there will be errors. Rules, which are usually written by many different people and change over a period of time, are very likely to suffer from errors. Syntax checkers can eliminate obvious errors, but there will be subtle errors in logic that are difficult to uncover. The clinical event monitor does not operate on its own, but is linked to a complex environment. As other clinical systems change their vocabulary, schema, data types, and event definitions, errors will occur in rule processing.

Clearly, if there is a syntax error in a rule, it can be detected and fixed. A syntax checker can also issue warnings for constructs that may indicate an existing logical error or that may lead to a future error. Examples of warnings include variables that are bound to a value but are never used, inappropriate uses of data types, and queries for data that are not actually stored in the database (a code may exist, but the data are not really collected).

Once the rules are being executed several types of errors are possible. These run-time errors are generally recorded in an error log. At CPMC, the following general-purpose error fields are defined:

- error time—when the error was detected
- task number—operating system's unique process number
- system—Unix, CICS, etc.
- terminal id—if there is a user at a terminal
- project name—used to organize the error log into projects
- transaction id—name of the executable program
- source file—name of the source code file
- programming language—format of the source file
- error type—application, database (SQL, IMS, VSAM), programming language, or operating system
- error function—what subroutine was executing
- error code—main error message
- error modifier—error subtype
- error data—additional information about the error
- source position—where in the source file was the error detected

- program data—information meant to reflect the state of the application at the time of the error

The broadest run-time error is a failure of the event monitor as a whole. In this case, event notification messages are usually retained on queue until the event monitor can be fixed. There may be an error processing an event. If the problem is due to an invalid event notification message (due to an ancillary application error), then the message is logged and the next message on the queue is processed. If the monitor is unable to fire rules (for example, if the event-rule trigger table is unavailable), then the current event notification message is rolled back to the queue, an error record is written, and the system is stopped until the problem is fixed. Those applications that call the event monitor synchronously will need to do without the monitor or wait until it is fixed.

If a rule encounters a fatal error, then an error record is written and the rest of the rules pertinent to the triggering event are processed. This could cause a problem, however, if a rule depends on the simultaneous execution of other rules. In general, such linking is better made explicit rather than implicit through the event triggering.

When a rule fails, the following additional error information is logged at CPMC (in the program data field):

- medical record number of the patient
- event type
- event code
- event time
- primary time of data associated with the event
- key (accession number) of data associated with the event
- current rule (numeric identifier and name)
- position in the rule where the error was detected

Nonfatal rule errors are recorded in the error log. The fact that an error occurred must somehow be communicated to the rule, in case there is any processing that needs to occur. In the Arden Syntax, communication is accomplished with the null data type. Whenever some processing fails, the associated variable is assigned the null data type. The implication is that a variable's true value is unknown. For example, if the patient's birthdate is requested from the database and it has not been recorded, then the null value is placed in the data variable. The Arden Syntax's default three-state logic generally produces the desired behavior for null values. If an expression is a disjunction of criteria, and any of them is true, then the null criterion will be ignored. If the other criteria are false, then the expression will be null (unknown). Conjunctions are processed analogously.

Another option is to add a rule component designed specifically to handle errors, similar to programming languages' ''on error'' condition. This permits more explicit, and perhaps safer, handling of errors.

The difficult question is what constitutes a fatal versus nonfatal error. The most conservative approach is to consider all errors fatal, but this may be too limiting. The types of errors that CPMC now considers nonfatal are (note that

all such errors are still logged, reviewed, and fixed): the unavailability of an ancillary database or application; unexpected data from an ancillary database; and processing errors such as division by zero (for example, when the rule author mistakenly thought a database field could never be zero).

Some logical malfunctions will never be detected even by a conservative system, and this will lead to false positive and false negative alert messages. At CPMC, we rely on user comments to alert us of problems we have missed (*57*), and we track how often each rule sends messages with an automated statistical tracker (*39*). If a rule begins to generate an unusual number of messages (too many or too few), then the tracker sends an alert to the system administrator. The rule firing statistics are also reviewed manually.

As in other areas, prevention is often easier than cure. Adequate testing is encouraged for all rules.

Conclusion

Clinical event monitors are beginning to gain acceptance in the health care community, and many vendors and institutions are in the process of developing event monitors. A robust implementation is difficult but worth the effort. The lessons learned developing clinical event monitors at the CPMC and other institutions may save developers time and effort.

Acknowledgment

This work was supported by National Library of Medicine grant R29 LM05627 "Linking Knowledge-based Systems to Clinical Databases"; National Library of Medicine LM04419 "IAIMS"; and a grant from the International Business Machines Corporation.

References

1. Rind, D. M., Safran, C., Phillips, R. S., Wang, Q., Calkins, D. R., Delbanco, T. L., Bleich, H. L., and Slack, W. V. Effect of computer-based alerts on the treatment and outcomes of hospitalized patients. *Arch. Int. Med.* **154,** 1511 (1994).
2. Haug, P. J., Gardner, R. M., Tate, K. E., Evans, R. S., East, T. D., Kuperman, G., Pryor, T. A., Huff, S. M., and Warner, H. R. Decision support in medicine: examples from the HELP system. *Comput. Biomed. Res.* **27,** 396 (1994).
3. McDonald, C. J. "Action-Oriented Decisions in Ambulatory Medicine." Year Book Medical Publishers, Chicago, 1981.
4. Barnett, G. O., Winickoff, R., Dorsey, J. L., Morgan, M. M., and Lurie, R. S. Quality assurance through automated monitoring and concurrent feedback using a computer-based medical information system. *Med. Care* **16,** 962 (1978).
5. Tierney, W. M., Miller, M. E., Overhage, J. M., and McDonald, C. J. Physician inpatient order writing on microcomputer workstations: Effects on resource utilization. *JAMA* **269,** 379 (1993).
6. Pryor, T. A., Dupont, R., and Clay, J. A MLM based order entry system: the use of knowledge in a traditional HIS application. *In* "Proceedings of the Fourteenth Annual Symposium on Computer Applications in Medical Care" (R. A. Miller, Ed.), pp. 579–583. IEEE Computer Society Press, New York, 1990.
7. Starren, J. B., Hripcsak, G., Jordan, D., Allen, B., Weissman, C., and Clayton, P. D.

Encoding a post-operative coronary artery bypass surgery care plan in the Arden Syntax. *Comput. Biol. Med.* **24,** 411 (1994).

8. Jenders, R. A., Zatsman, P., Estey, G., and Barnett, G. O. Use of different techniques to implement health maintenance guidelines in the development of a clinical workstation. *In* "Proceedings of the 1993 Spring Congress of the American Medical Informatics Association," p. 116. American Medical Informatics Association, Bethesda, MD, 1993.
9. Warner, H. R., Olmstead, C. M., and Rutherford, B. D. HELP—A program for medical decision-making. *Comput. Biomed. Res.* **5,** 65 (1972).
10. Hripcsak, G., Cimino, J. J., Johnson, S. B., and Clayton, P. D. The Columbia-Presbyterian Medical Center decision-support system as a model for implementing the Arden Syntax. *In* "Proceedings of the Fifteenth Annual Symposium on Computer Applications in Medical Care" (P. D. Clayton, Ed.), pp. 248–252. McGraw–Hill, New York, 1992.
11. American Society for Testing and Materials. E 1460 Standard Specification for Defining And Sharing Modular Health Knowledge Bases (Arden Syntax for Medical Logic Modules). *In* "1992 Annual Book of ASTM Standards," Vol. 14.01, pp. 539–587. American Society for Testing and Materials, Philadephia, 1992.
12. Magyar, G., Arkad, K., Ericsson, K. E., Gill, H., Gao, X., Linnarsson, R., and Wigertz, O. B. Realizing medical knowledge in MLM form as working modules in a patient information system. *In* "Software Engineering in Medical Informatics" (T. Timmers and B. I. Blum, Eds.), pp. 481–498. North-Holland, Amsterdam, 1991.
13. Lepage, E., Traineau, R., Marchetti, P. H., Benbunan, M., and Gardner, R. M. Development of a computerized knowledge based system integrated to a medical workstation: application to blood transfusion. *In* "Proceedings of the Seventh World Congress on Medical Informatics (MEDINFO 92)" (K. C. Lun *et al.,* Eds.), pp. 585–590. North-Holland, Amsterdam, 1992.
14. Dupuits, F. M., and Hasman, A. HIOS+, a decision aid in medicine. *In* "Proceedings of the Seventh World Congress on Medical Informatics (MEDINFO 92)" (K. C. Lun *et al.,* Eds.), pp. 454–460. North-Holland, Amsterdam, 1992.
15. Campbell, J. R., Stoupa, R., and Warren, J. J. Design and implementation of a rule based system for ambulatory nursing data management. *In* "Proceedings of the Fifteenth Annual Symposium on Computer Applications in Medical Care" (P. D. Clayton, Ed.), pp. 659–663. McGraw–Hill, New York, 1992.
16. Rankin, T. L. "Artificial Intelligence and Health Industry Marketing" (TR52.0031). IBM, Atlanta, 1991.
17. Spates, R. P., and Aller, K. C. One vendor's experience: Preliminary development of a reminder system based on the Arden Syntax. *Comput. Biol. Med.* **24,** 371 (1994).
18. Ludemann, P. Experience with the Arden Syntax with a clinical event monitor and other systems. *Comput. Biol. Med.* **24,** 377 (1994).
19. Kuhn, R. A., and Reider, R. S. An approach to developing a compiler for Arden Syntax using a C++ application framework. *Comput. Biol. Med.* **24,** 365 (1994).
20. Pryor, T. A., Gardner, R. M., Clayton, P. D., and Warner, H. R. The HELP system. *In* "Information Systems For Patient Care" (B. I. Blum, Ed.), pp. 109–128. Springer-Verlag, New York, 1984.
21. McDonald, C. J., Blevins, L., Tierney, W. M., and Martin, D. K. The Regenstrief Medical Records. *MD Comput.* **5**(5), 34–47 (1988).
22. Musen, M. A. Dimensions of knowledge sharing and reuse. *Comput. Biomed. Res.* **25,** 435 (1992).
23. Shwe, M. A., Sujansky, W., and Middleton, B. Resue of knowledge represented in the Arden Syntax. *In* "Proceedings of the Sixteenth Annual Symposium on Computer Applications in Medical Care" (M. E. Frisse, Ed.), pp. 47–51. McGraw–Hill, New York, 1993.
24. Gao, X., Shahsavar, N., Arkad, K., Ahlfeldt, H., Hripcsak, G., and Wigertz, O. Design and functions of medical knowledge editors for the Arden Syntax. *In* "Proceedings of the Seventh World Congress on Medical Informatics (MEDINFO 92)" (K. C. Lun *et al.,* Eds.), pp. 472–477. North-Holland, Amsterdam, 1992.
25. Shabot, M. M., LoBue, M., Leyerle, B. J., and Dubin, S. B. Inferencing strategies for automated ALERTS on critically abnormal laboratory and blood gas data. *In* "Proceedings of the Thirteenth

Annual Symposium on Computer Applications in Medical Care'' (L. C. Kingsland, Ed.), pp. 54–57. IEEE Computer Society Press, New York, 1989.

26. VAN DER LEI, J., AND MUSEN, M. A. Separation of critiquing knowledge from medical knowledge: implications for the Arden Syntax. *In* ''Software Engineering in Medical Informatics'' (T. Timmers and B. I. Blum, Eds.), pp. 499–509. North-Holland, Amsterdam, 1991.
27. PROKOSCH, H. U., KAMM, S., WIECZOREK, D, AND DUDECK, J. Knowledge representation in the pharmacology: A possible application area for the Arden Syntax? *In* ''Proceedings of the Fifteenth Annual Symposium on Computer Applications in Medical Care'' (P. D. Clayton, Ed.), pp. 243–247. McGraw–Hill, New York, 1992.
28. JOHANSSON, B. G., AND WIGERTZ, O. B. An object oriented approach to interpret medical knowledge based on the Arden Syntax. *In* ''Proceedings of the Sixteenth Annual Symposium on Computer Applications in Medical Care'' (M. E. Frisse, Ed.), pp. 52–56. McGraw–Hill, New York, 1993.
29. JOHNSON, B., MCNAIR, D., KAILASAM, K., REILLY, R., EKLUND, N., MCCOY, G., AND JAMIESON, P. Discern—An integrated prospective decision support system. In ''Proceedings of the Eighteenth Annual Symposium on Computer Applications in Medical Care'' (J. G. Ozbolt, Ed.), p. 969. Hanley & Belfus, Inc., Philadelphia, 1994.
30. HAIMOWITZ, I. J. AND KOHANE, I. S. Influences on the performance of hospital clinical event monitoring. *In* ''Proceedings of the Fifteenth Annual Symposium on Computer Applications in Medical Care'' (P. D. Clayton, Ed.), pp. 614–618. McGraw–Hill, New York, 1992.
31. KOHANE, I. S. Maintaining alternate interpretations of data from multiple sources in a clinical event monitoring system. *In* ''Proceedings of the Seventh World Congress on Medical Informatics (MEDINFO 92)'' (K. C. Lun *et al.,* Eds.), pp. 483–489. North-Holland, Amsterdam, 1992.
32. WAGNER, M. M., AND COOPER, G. F. Decision-theoretic information pretrieval: A generalization of reminding. *In* ''Proceedings of the Seventeenth Annual Symposium on Computer Applications in Medical Care'' (C. Safran, Ed.), pp. 512–516. McGraw–Hill, New York, 1994.
33. BANKS, N. J., ROSE, P. G., AND PALMER, R. H. Prompt for COSTAR—A clinical reminder system. *In* ''Proceedings of the Thirteenth Annual Symposium on Computer Applications in Medical Care'' (L. C. Kingsland, Ed.), pp. 941–942. IEEE Computer Society Press, New York, 1989.
34. VAN DER LEI, J., AND MUSEN, M. A. A model for critiquing based on automated medical records. *Comput. Biomed. Res.* **24,** 344 (1991).
35. FORDHAM, D., MCPHEE, S. J., BIRD, J. A., RODNICK, J. E., AND DETMER, W. M. The cancer prevention reminder system. *MD Comput.* **7,** 289 (1990).
36. BARTON, M. B., AND SCHOENBAUM, S. C. Improving influenza vaccination performance in an HMO setting: The use of computer-generated reminders and peer comparison feedback. *Am. J. Public Health* **80,** 534 (1990).
37. KAHN, M. G., STEIB, S. A., FRASER, V. J., AND DUNAGAN, W. C. An expert system for culture-based infection control surveillance. *In* ''Proceedings of the Seventeenth Annual Symposium on Computer Applications in Medical Care'' (C. Safran, Ed.), pp. 171–175. McGraw–Hill, New York, 1994.
38. JOHNSTON, M. E., LANGTON, K. B., HAYNES, B., AND MATHIEU, A. Effects of computer-based clinical decision support system on clinical performance and patient outcome. *Ann. Intern. Med.* **120,** 135 (1994).
39. HRIPCSAK, G. Monitoring the Monitor: Automated Statistical Tracking of a Clinical Event Monitor. *Comput. Biomed. Res.* **26,** 449 (1993).
40. MCCARTHY, D. R., AND DAYAL, U. The architecture of an active data base management system. *In* ''Proceedings of the 1989 ACM SIGMOD International Conference on the Management of Data'' (J. Clifford and B. Lindsay, Eds.), pp. 215–224. ACM Press, 1989.
41. DAYAL, U. Active database management systems. *In* ''Proceedings of the Third International Conference on Data and Knowledge Bases,'' pp. 150–169. IPA, Israel, 1988.
42. COHEN, D. Compiling complex database transition triggers. *In* ''Proceedings of the 1989 ACM SIGMOD International Conference on the Management of Data'' (J. Clifford and B. Lindsay, Eds.), pp. 225–234. ACM Press, 1989.

43. "HELP Frame Manual, Version 1.6." LDS Hospital, Salt Lake City, 1989.
44. HRIPCSAK, G., LUDEMANN, P., PRYOR, T. A., WIGERTZ, O. B., AND CLAYTON, P. D. Rationale for the Arden Syntax. *Comput. Biomed. Res.* **27,** 291 (1994).
45. HRIPCSAK, G. Writing Arden Syntax medical logic modules. *Comput. Biol. Med.* **24,** 331 (1994).
46. NECHES, R., FIKES, R., FININ, T., GRUBER, T., PATIL, R., SENATOR, T., AND SWARTOUT, W. R. Enabling technology for knowledge sharing. *AI Magazine* **12**(3), 36 (1989).
47. SYBASE, INC. User Manual. Emeryville, CA: Sybase, Inc., 1992.
48. LINDBERG, D. A. B., and HUMPHREYS, B. L. Toward a unified medical language. *In* "Proceedings of the Seventh International Medical Informatics Congress," pp. 23–31. Springer-Verlag, Germany, 1987.
49. NATIONAL LIBRARY OF MEDICINE. "UMLS Knowledge Sources, Second Experimental Edition." National Library of Medicine, Bethesda, MD, 1991.
50. CIMINO, J. J., HRIPCSAK, G., JOHNSON, S. B., AND CLAYTON, P. D. Designing an Introspective, Multipurpose, Controlled Medical Vocabulary. *In* "Proceedings of the Thirteenth Annual Symposium on Computer Applications in Medical Care" (L. C. Kingsland, Ed.), pp. 513–518. IEEE Computer Society Press, New York, 1989.
51. JOHNSON, S. B., HRIPCSAK, G., CHEN, J., AND CLAYTON, P. D. Accessing the Columbia Clinical Repository. *In* "Proceedings of the Eighteenth Annual Symposium on Computer Applications in Medical Care" (J. G. Ozbolt, Ed.), pp. 281–285. Hanley & Belfus, Inc., Philadelphia, 1994.
52. HRIPCSAK, G., JOHNSON, S. B, AND CLAYTON, P. D. Desperately seeking data: knowledge base-database links. *In* "Proceedings of the Seventeenth Annual Symposium on Computer Applications in Medical Care" (C. Safran, Ed.), pp. 639–643. McGraw–Hill, New York, 1994.
53. WIEDERHOLD, G., JAJODIA, S., AND LITWIN, W. "Dealing with Granularity of Time in Temporal Databases." Stanford University, Stanford, CA, 1990.
54. KAHN, M. G., TU, S., AND FAGAN, L. M. TQuery: A context-sensitive temporal query language. *Comput. Biomed. Res.* **24,** 401 (1991).
55. DAS, A. K., TU, S. W., PURCELL, G. P., AND MUSEN, M. A. An extended SQL for temporal data management in clinical decision-support systems. *In* "Proceedings of the Sixteenth Annual Symposium on Computer Applications in Medical Care" (M. E. Frisse, Ed.), pp. 128–132. McGraw–Hill, New York, 1993.
56. OSTLER, M. R., STANSFIELD, J. D., AND PRYOR, T. A. A new, efficient version of HELP. *In* "Proceedings of the Ninth Annual Symposium on Computer Applications in Medical Care" (M. J. Ackerman, Ed.), pp. 296–297. IEEE Computer Society Press, New York, 1985.
57. HRIPCSAK, G., AND CLAYTON, P. D. User comments on a clinical event monitor. *In* "Proceedings of the Eighteenth Annual Symposium on Computer Applications in Medical Care" (J. G. Ozbolt, Ed.), pp. 636–640. Hanley & Belfus, Inc., Philadelphia, 1994.