Design of A Clinical Alert System to Facilitate Development, Testing, Maintenance, and User-Specific Notification

Michael I. Oppenheim, MD, Ronald J. Mintz, PhD, Aurelia G. Boyer, MSN,MBA, William W. Frayer, MD
Weill Medical College of Cornell University
NewYork Weill Cornell Medical Center, NewYork-Presbyterian Hospital
New York, N.Y.

Creation and maintenance of electronic clinical alerts within a hospital's electronic medical record (EMR) or database poses a number of challenges. Development can require significant programming effort. Final testing should ideally be performed in a real clinical environment without clinician notification, which may create technical challenges. After an alert is in production, modifications may become necessary in response clinician feedback, changes in clinical factors, or technical issues. Changes may be required in the knowledge base utilized by the alert or in the presentation of the alert condition to the clinicians. Occasionally, different users within the clinical environment may wish to have the same alert data presented differently. We have developed a strategy which allows development of multi-functional alerts and facilitates modification of alert function and/or presentation with minimal to no programming effort. Some elements of this scheme may be appropriate for incorporation into clinical alerting standards.

INTRODUCTION

Clinical Alerting is rapidly becoming an important feature in hospital EMRs and databases. They have been shown to improve the quality[1,2] and potentially the cost[3] of medical care delivery. However, the development, maintenance, and notification management pose significant challenges.

Because of the programming effort and medical domain knowledge required, development of alerts can be a resource intensive process. This has prompted the development of the Arden Syntax[4] as an attempt to allow less technical personnel to create alerts and to facilitate sharing of alerts. However, the syntax was not specifically designed with *reuse*-ability as a goal and the development of multiple similar alerts is not specifically facilitated[4]. Kuperman et al. recognized the relatively generic nature of the logic component of many different types of alerts, and described an elegant strategy for rapidly developing multiple similar-type alerts[5]. However, this produces a large library of alerts to manage.

Testing and deployment of clinical alerts poses some complex challenges. The clinical staff working in different parts of a hospital or in different outpatient areas may have different clinical expertise and experience. Alerts relevant to some may be unnecessary or even disruptive to others. For example, we have an alert suggesting pregnancy screening in gestational-age women when radiologic studies are ordered (if no pregnancy test is found in the laboratory results). This is appropriate in general medical or surgical environments, but would likely prove disruptive in an Obstetric/Gynecologic specialty environment. Moreover, in addition to simple "on/off" control, we wanted the ability to have the same alert in clinical production in some areas of the hospital but executing in a testing mode in others. Conventional triggering mechanisms do not provide this level of flexibility.

Clinical alerts in production may require maintenance over time. As clinical knowledge changes, alerts must accommodate those changes. For example, we have an alert to remind clinicians to discontinue or dose-adjust hypoglycemia-inducing medications when a patient's oral intake is suspended (NPO). As new diabetes medications are developed and marketed, the alert needs to be updated to recognize these drugs. Changes in configurations of hospital systems also may necessitate modifications to existing alerts[6]. In our environment, a change from one laboratory reporting system to a different system (which uses different test codes) required modification of certain laboratory value based alerts. In an environment where many alerts exist, maintenance can be daunting task[7,8].

User notification of alert conditions presents a unique set of challenges. Our Clinical Alerts Committee, comprised of representatives of different user classes (attending physicians, housestaff physicians, nurses, technical personnel, ancillary services personnel), developed a series of requirements:

1067—5027/00/$5.00

(1) Each alert should be visible only to the user classe(s) for which it is appropriate
(2) Acknowledgment privileges should be independent of viewing privileges (i.e. there can be user classes which have view-only privileges)
(3) The same alert condition can be presented differently to different user classes (e.g. a critical value might be presented to an attending physician simply as a critical laboratory result, while medical students or housestaff might also see information regarding etiology/management)
(4) Critical alerts should be displayed by means of popup dialog boxes that force the user to view prior to proceeding. However, less critical information can be conveyed through display in a (passive) message box on the screen.

The NYWCMC EMR

The electronic medical record used at the NewYork Weill Cornell Medical Center, a tertiary care academic medical center, is a commercial product not explicitly designed to support clinical alerting functionality. It is a C-language based and executes on a UNIX platform in a distributed fashion. The software is compiled on a single server and distributed to multiple servers across the hospital. The alerts themselves are written as C code modules whose execution is triggered in response to user or electronic (e.g. interface) database activity.

This system architecture creates issues in addition to the generic issues described above. Most significant is that development or modification of existing alerts requires recompilation and distribution of the new software. After distribution, the old software must be stopped and the new program started, creating undesirable workflow interruptions.

This paper describes our effort to address all of the aforementioned issues and design requirements. We set out to develop a clinical alerting system based on the following design principles:

- Develop clinical alerts generically; similar tasks should be handled using a single multifunctional code module
- Externalize the knowledge to allow generic alert generation and facilitate knowledge base changes

Test	Sex	Abs_hi	Abs_low	%_inc	%_dec
K	*	5.8	3.5	N/A	N/A
Hgb	M	N/A	8.0	N/A	10
Hgb	F	N/A	6.0	N/A	10
Cr	*	8.0	N/A	15	N/A
PTT	*	100.0	N/A	N/A	N/A

Figure 1: Critical lab value table: Indicates the absolute high and low values and percentage change thresholds for alerting

- Manage execution locations and execution modes (e.g. testing vs. production) without requiring recompilation
- Customize presentation of alerts (what message, who sees, who acknowledges, etc.) to different user classes

SYSTEM DESIGN

We utilize a strategy of run time database lookups (during alert execution) to control functionality. Alert behavior is altered by making changes to tables rather than by changing the code modules themselves. We have applied this methodology to address each of the design principles.

Generic Alert Code/Externalization of Content:

Nearly all of our clinical alerts modules read some element(s) of their domain knowledge at run time, rather than contain hard-coded values. For example, rather than having an alert for high potassium and a separate alert for low hemoglobin, a single critical value alert module was developed which reads a database table at run time (Figure 1).

Using this strategy, a single code module can generate critical lab value alerts for as many lab tests as desired. A single alert module can have multiple triggers defined; at the time the alert is triggered, it is passed context information about the triggering event. The alert code can then look up the critical thresholds based on the specific lab test that triggered it to execute. The alert code itself requires no hard-coded domain knowledge about specific laboratory tests. This minimizes the number of alert modules required to perform multiple similar type tasks.

In addition to simplifying development, this strategy also provides important maintenance benefits. For example, for our NPO/Hypoglycemic drug alert, simply adding the names of new anti-diabetes medications to a table allows the alert to recognize the new medication during subsequent executions.

The use of run-time lookups to control alerting functionality is certainly not new. This strategy is frequently employed for areas where knowledge content is purchased and updated frequently, such as drug interaction checking. Extensions to the Arden Syntax for this purpose have been described[9].

Execution Control:

Our execution control table allows control based on either the geographic unit on which the patient is located, the specialty service to which the patient is assigned, or a combination of both.

At the start of execution, each code module calls a function which returns two values. The first is a boolean value indicating whether the alert should execute at all. A return value of FALSE indicates that the alert is not appropriate for the patient in whose record it was triggered. The second return value is a coded value indicating the mode in which the rule should execute (production vs. testing). The execution mode parameter is similar to the validation slot of the Arden Syntax, except that the information is external to the alert module. It should be noted that it is the responsibility of the alert developer to use the returned value. The alert may be written with both the testing and production functionality in the same module, and execution controlled by means of:

```
switch(execmode):
{
case(TESTING): {testing code}
case(PRODUCTION): {production code}
default: {table error}
}
```

Additionally, the execution mode is used as a parameter to the notification routines.

Use of an execution control table allows us to limit execution to the units and/or services desired. Additionally, execution mode can be different for different locations/specialties. Most importantly, an alert can be activated/deactivated or its mode of execution changed through the modification of this table without recompilation of software.

Custom User Class Notification:

Our user-notification table is demonstrated in figure 2. Note that, in the example given, the message given to medical students regarding a critical laboratory value (`crit_lab`) is different than the message given to physicians. Notification is deliberately suppressed for all other user classes (based on the `display_mode` field).

Alerts generated in response to user activity (e.g. order entry) are reported immediately. Alerts triggered by other events (e.g. lab data) create an entry in the patient database indicating the situation detected. At the next "chart open", the message text and reporting mode is determined based on the situation and the user class of the clinician opening the chart, and the alert is displayed (if appropriate for that clinician's user class). The `display_mode` field of the messaging table indicates whether the user can acknowledge the alert (presented as an option within the dialog box) or whether they do not have rights to acknowledge the alert. In the example shown, a student would see the alert regarding a critical lab value, but is not able to acknowledge it (`noack`). As a result, the alert would be displayed (to other users opening the chart - a user is not shown the same alert more than once) until a physician acknowledges the condition.

A single alert may have more than one set of messages associated with it. Consider our radiology-pregnancy screening alert: if a patient has previous radiology orders in which the pregnancy field was marked "no", our alert assumes the patient is not pregnant. Conversely, a previous radiology order might indicate that the patient is, in fact, pregnant. A message regarding the former is safe to display (passively) in the screen's message box, while the latter should create a dialog box interrupting the clinician's workflow to indicate this important piece of data. Therefore, lookups within this table can be based on a specific `situation` within a clinical alert module, allowing a single alert module to generate different messages for different situations. For each situation, the text and reporting mode can be specific to different user classes.

Because this table is read at run time, modification of this table alters message text and reporting mode without requiring recompilation.

RESULTS

Using the strategy described above, we have successfully developed and implemented clinical alerting functionality within our EMR. At the time of this writing, 4 alerts are in clinical production, 1 is executing in testing mode, 2 are ready for testing, and

Situation	user_class	message	display_mode
crit_lab	attend_md	Warning: <#test> = <#result>	popup_noack
crit_lab	resident_md	Warning: <#test> = <#result>	popup_ack
crit_lab	med_student	Warning: <#test> = <#result>. Contact responsible physician immediately	popup_noack
crit_lab	*	NO MESSAGE	suppress
rad_prevno	*	Note: Not pregnant status being assumed from prior radiology order	message_box
rad_prevpreg	*	Warning: Patient charted as pregnant in previous radiology order	popup_ack

Figure 2: Sample messaging table: see text for discussion; <#xxxx> indicates a parameter added to the message during display

2 are under development. We are utilizing all of the features described except for user-class specific messaging; this feature is utilized by the alert currently in testing. 3 units (60 beds) are using full electronic order entry; for those units, our system generates 2-3 alerts/unit/day related to pharmacy order entry. The remainder of the hospital (765 beds) has limited electronic order entry; only our radiology-pregnancy screening reminder is relevant.

Because of the limited time since our implementation, we have not yet encountered many changes to the clinical components of our knowledge base. However, we have undergone system configuration changes, and adjustment of alerts to accommodate these changes has been greatly facilitated. Newly created rules are first tested by executing in testing mode, then are put into production on a single unit, and finally full production mode. This process has been greatly simplified by having the ability to control location and mode of execution without recompilation.

To date, we have seen minimal performance impact as a result of implementation of alerts. Because our strategy is less computationally efficient (see Discussion), close monitoring is required as the alert library expands.

DISCUSSION

In some ways, our approach deviates from the strategy supported by the Arden Syntax. The Arden Syntax specifically does not separate domain knowledge (medical facts about the problem being addressed) from problem-solving knowledge (how to use facts to perform an intelligent action)[4]. It is known that the logic slot of MLMs is the most likely slot to change over time[7], though it is not known which components of the logic slot are most often altered. Our underlying assumption is that domain knowledge is dynamic (e.g. new drugs, new thresholds, newly reported interactions, etc.), while problem-solving knowledge is relatively constant. Therefore, we have externalized most of the domain knowledge and developed generic alert modules with hard-coded problem-solving knowledge.

Kuperman et. al. describe the use of generic boolean conditional primitives which are instantiated with specific clinical parameters to create unique alerts[5]. In effect, we have employed a very similar strategy; however, rather than instantiating to build multiple unique alerts, the instantiation (or domain knowledge) is supplied at run-time by means of database lookups which supplies the appropriate clinical parameters to the generic conditional (the problem solving knowledge) encoded in the alert module.

The system we have developed provides a number of benefits. First, our strategy allows our alert library to remain smaller, which facilitates maintenance. Second, consolidation of the knowledge base facilitates implementation of changes and assures consistency where domain knowledge is utilized by multiple different alerts. Third, our strategy provides us with the ability to manage testing, deployment, and post-production modifications of the knowledge-base/content of our clinical alerts with minimal impact on system function. Finally, we have the ability to customize alert condition reporting.

The problems of knowledge base changes are of two types: changes in clinical knowledge and changes in system design/configuration. Changes in clinical knowledge present a special difficulty because the technical experts may not be clinical domain experts, and the domain experts may not be aware of the alert library and the impact of new clinical knowledge. This problem must be solved via a process solution, not a technical one[8]. However, while our strategy cannot assist with *identification* of elements requiring upgrade because of changes in domain knowledge, the *implementation* of such changes is greatly facilitated by our table-driven design. In contrast to domain knowledge changes, identification and implementation of alert maintenance necessitated by system design or configuration changes is greatly facilitated when domain knowledge is externalized. In our case, a change in our laboratory reporting system led to a change in laboratory value representation within the EMR database. This was handled by identifying the run-time table(s) containing laboratory dependencies. Rather than having to identify and update a hyperkalemia alert, an anemia alert, an over-anticoagulation alert, and etc., identification and modification of a small number of tables was sufficient.

Drug	Lab	dir	% chng	Abs val
Famotidine	Plt	<	20	90.0
Captopril	K	>	N/A	4.5
Gancyclovir	WBC	<	50	1.5

Figure 3: Drug-lab interaction table: maps drugs to specific laboratory tests and laboratory abnormalities (absolute values or relative changes) considered significant

Our strategy of using run-time lookups is not limited only to tasks requiring simple data lookups; more complex relationships can be handled in this manner as well. For example, we plan to automate adverse drug event (ADE) detection by developing a table which maps pharmaceutical agents

to laboratory tests and threshold values, as depicted in figure 3. A single alert module would be able to recognize multiple ADEs. If a triggering laboratory value exceeds (or fails to reach) the associated threshold value indicated in the table, the alert can determine all of the drugs associated with that laboratory abnormality and cross check the patient's medication list.

The most significant disadvantage of this approach is the computational expense. During execution, multiple database queries are required to control function. At a minimum, all alerts dynamically determine their execution status and mode (production / testing), and the alert message and notification method. Additionally, context-specific domain knowledge requires extra table lookups. Finally, our execution control method does not actually limit the execution of alert code to specified patients; rather, the alert code executes and self-terminates if appropriate. While we recognize that this wastes computational resources, our system provides no mechanism to limit the *triggering* of an alert to a subset of patients.

Another disadvantage to our approach is that it requires specific expertise during alert development. For example, the onus is on the programmer to call the execution control functions, and to handle the execution mode parameter properly. Conversely, notification is greatly facilitated with our strategy. Our notification functions receive (a) a code for the situation to report, (b) the user class, and (c) the execution mode as parameters and handle the details of the reporting based on the messaging table.

FUTURE DIRECTIONS

We believe that the use of run-time controls provides a greater level of flexibility (for maintenance) and control (during execution) of clinical alerts. We believe that some of these principles could be incorporated within the Arden Syntax without corruption of the standardization ideal. For example, the evoke slot syntax could be modified to accept qualifying statements. If context passing were implemented to provide an MLM with information about the triggering event, that information could be used in a standardized `read` construct designed to retrieve domain-specific knowledge at run time. It is our hope that some of these strategies will eventually be incorporated into the Arden Syntax.

CONCLUSION

We have developed a strategy which allows us to develop multi-functional clinical alerts within our electronic medical record. The execution, function, and notification of our alerts is determined by the run-time environment. This approach is necessary in computing environments where recompilation of alert modules is not feasible. The technique provides significant advantages that are likely to be applicable to other systems and environments.

ACKNOWLEDGEMENTS

We are indebted to Dr. John Brimm for his support (both technical and non-technical) during our development efforts.

REFERENCES

(1) Raschke RA, Gollihare B, Wunderlich TA et al. A computer alert system to prevent injury from adverse drug events: development and evaluation in a community teaching hospital. JAMA 1998; 280(15):1317-1320.

(2) Haug PJ, Gardner RM, Tate KE et al. Decision support in medicine: examples from the HELP system. Comput Biomed Res 1994; 27(5):396-418.

(3) Evans RS, Larsen RA, Burke JP et al. Computer surveillance of hospital-acquired infections and antibiotic use. JAMA 1986; 256(8):1007-1011.

(4) Hripcsak G, Ludemann P, Pryor TA, Wigertz OB, Clayton PD. Rationale for the Arden Syntax. Comput Biomed Res 1994; 27(4):291-324.

(5) Kuperman GJ, Teich JM, Bates DW, McLatchey J, Hoff TG. Representing hospital events as complex conditionals. Proc Annu Symp Comput Appl Med Care 1995;137-141.

(6) Hripcsak G. Monitoring the monitor: automated statistical tracking of a clinical event monitor. Comput Biomed Res 1993; 26(5):449-466.

(7) Jenders RA, Huang H, Hripcsak G, Clayton PD. Evolution of a knowledge base for a clinical decision support system encoded in the Arden Syntax. Proc AMIA Symp 1998;558-562.

(8) Kuperman GJ, Fiskio JM, Karson A. A process to maintain the quality of a computerized knowledge base. Proc AMIA Symp 1999;87-91.

(9) Sager J.T., Corman R.G., Daine D.N., Wu J, Yang M.G., Thalman N et al. The Transfer of Adverse Clinical Events Rules with an Arden-Based Decision Support System. Proc AMIA Symp 1997;995.