MEDINFO 2015: eHealth-enabled Health
I.N. Sarkar et al. (Eds.)
© 2015 IMIA and IOS Press.
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License.
doi:10.3233/978-1-61499-564-7-285

INITIATE: An Intelligent Adaptive Alert Environment

Borna Jafarpour[a], Samina Raza Abidi[b], Ahmad Marwan Ahmad[a], Syed Sibte Raza Abidi[a]

[a] *NICHE Research Group, Faculty of Computer Science, Dalhousie University, Halifax, Canada*
[b] *Medical Informatics, Faculty of Medicine, Dalhousie University, Halifax, Canada*

Abstract

Exposure to a large volume of alerts generated by medical Alert Generating Systems (AGS) such as drug-drug interaction softwares or clinical decision support systems overwhelms users and causes alert fatigue in them. Some of alert fatigue effects are ignoring crucial alerts and longer response times. A common approach to avoid alert fatigue is to devise mechanisms in AGS to stop them from generating alerts that are deemed irrelevant. In this paper, we present a novel framework called INITIATE: an INtellIgent adapTIve AlerT Environment to avoid alert fatigue by managing alerts generated by one or more AGS. We have identified and categories the lifecycle of different alerts and have developed alert management logic as per the alerts' lifecycle. Our framework incorporates an ontology that represents the alert management strategy and an alert management engine that executes this strategy. Our alert management framework offers the following features: (1) Adaptability based on users' feedback; (2) Personalization and aggregation of messages; and (3) Connection to Electronic Medical Records by implementing a HL7 Clinical Document Architecture parser.

Keywords:

Alert Fatigue; Alert Management Engine; Alert Management Strategy; Semantic Web; HL7-CDA.

Introduction

In recent years, medical Alert Generating Systems (AGS), such as vital signs monitoring devices, drug-drug interaction systems, and clinical decision support systems, have become prevalent in health-care environments. Extensive exposure to a large volume of alerts, especially irrelevant alerts, overwhelms users of these systems and causes alert fatigue. Alert fatigue leads to ignoring vital alerts, longer response times, anxiety in healthcare professionals and incorrect decisions [1]. Due to these undesirable effects, several attempts have been made to address the issue of alert fatigue. Existing approaches can be classified under two general categories: (a) *Suppressing alerts*: the rules generating alerts in AGS are modified so that irrelevant alerts are generated less frequently. For instance, in drug-drug interaction systems, if route of administration is considered in the head of rule, several alerts pertaining to nonexistent interactions will be avoided. In this approach, since the alert management strategy is implemented in the AGS, it cannot be reused in other AGS. Moreover, this approach is not readily applicable to situations where several AGS are operating concurrently; and (b) *Managing alerts*: in this approach, an alert management strategy is defined for each type of alert. An alert management engine monitors the generated alerts by the AGS and manages their lifecycles based on their management strategy. Hence, alert fatigue is prevented by less frequent and smarter generation of alerts as they can only be raised when they are in critical stages of their lifecycle and certain conditions are met. Since alert management is performed externally, this alert management strategy can scaled across multiple concurrent AGS.

In this paper, we present a novel alert management framework featuring an *Alert Management Strategy Language (AMSL)* represented as a Web Ontology Language (OWL) ontology; and, coupled with an *Alert Management Engine (AME)* that executes the alert management strategy based on patient information accessible from a *Health Level 7* (HL7) compliant Electronic Medical Records (EMR) using the *Clinical Document Architecture* (HL7-CDA [2]). We have identified and categorized the lifecycle of different alerts and have developed alert management logic as per the alerts' lifecycle. Our framework implements a set of unique features to address alerts fatigue, such as a smart alert counter that filters out irrelevant alerts based on their time stamps, alert notification adaptability based on users' feedback, aggregation of alerts to a unified alert, responding to the alert delivery medium and possibility of connecting to commercial EMR systems. We leverage semantic web technologies to represent the alerts and to manage them.

Related Work

Approaches to avoid alert fatigue can be categorized into three categories. In the first approach, the underlying rules generating the alerts are modified to limit the generation of irrelevant/repetitive alerts or to delay the notification of the alerts until a point that the alert becomes critical. For instance, in vital signs monitoring devices, generation of an alert regarding high heart rate can be delayed until a certain amount of time has passed since the heart rate has been more than a predefined threshold [3]. In another example, *Rule1* that represents theophylline–cimetidine interaction in a drug-drug interaction system is modified in terms of *Modified_Rule1* so that route of administration is taken into consideration to make the rule fire less frequently and more accurately [4]:

Rule1: If (theophylline and cimetidine)$\rightarrow$ Alert (Name: theophylline –cimetidine, Message: "….")

Modified_Rule1: If (theophylline and (**oral** cimetidine)$\rightarrow$ Alert (Name: theophylline –cimetidine, Message: "…")

The approach to modify alert generation rules to address alert fatigue has the following limitations: (1) the alert generation logic is encoded within the rules, and hence, modifications to the rules is local to the AGS and it cannot be reused by other AGS; and (2) if several sources of alerts exist, each AGS rule set needs to be modified which may eventually lead to inconsistencies across the multiple alert generation rules.

The second approach to address alert fatigue is to allow the AGS to generate the alerts, but subsequently filter the irrelevant alerts using an auxiliary alert management engine. To filter irrelevant alerts, domain experts' knowledge regarding relevancy of alerts is captured in terms of a classifier. In this approach, AGS users tag the alerts, as either relevant or irrelevant, in a normal no alert fatigue prevention setting. These tags are, then, used by machine learning algorithms to train a classifier to determine if an alert should be suppressed or presented to the user [5]. Disadvantage of this approach is the need for a large vole of training data that cover the entire breath of the alerts in different clinical scenarios, and the fact that the decision logic of such classifiers cannot be interpreted by healthcare professionals.

The third approach to avoid alert fatigue is to capture domain experts' knowledge in terms of an alert management language and represent the lifecycle of an alert—i.e. describing when and how often an alert should be shown to the user. Klimov et al. [6] have defined a comprehensive language to define alert lifecycles.

In our work, we pursue the third approach as it allows an explicit description of alerts and their lifecycles, which can be used to develop generic AGS. In comparison to Klimov et al. [6], our approach is different in the following ways: we define the concept of a *counter* that filters the alerts based on their currency—i.e. either the alerts are deemed too old to be relevant or are regarded as too soon to the previous alert in order for it to be meaningful. We also enhanced our framework by adding a set of desirable features such as adaptability based on user feedback, aggregation of messages, consistent alert management framework across several AGS operating concurrently, and connecting to EMR to be able to use patient information in alert lifecycle definition.

INITIATE: an INtellIgent adapTIve AlerT Environment

Our alert management framework shown in Figure 1 comprises the following components:

(1) An OWL ontology that represent our Alert Management Strategy Language (AMSL).

(2) Alert Management Engine (AME) that exposes RESTful web services for AGS to submit their generated alerts. This engine manages alerts based on the alert management knowledge encapsulated in an instantiation of the AMSL. To interpret the expressions that make use of lab test result and medication conditions, this engine contains an HL7-CDA parser that evaluates validity of expressions such as *allopurinol on hold* and *Synthroid dosage > 100mg*, which are representable in AMSL.

(3) CouchDB that is a NoSQL (JSON document based) database storing the following items:

a. HL7-CDA documents each representing a patient medical record.

b. Instantiation of the AMSL ontology stored as a RDF/JSON document that represents the lifecycle of alerts generated by the connected AGS.

c. Alert Status: each alert status represents how many times and when a specific alert has been generated for a patient. We will see an example of this later in the paper.

We used a document-based database to avoid normalizing our ontology and HL7-CDA documents into a relational table in order to store the documents as a whole.

Alert Management Strategy Language

In this section, we discuss AMSL, which is formalized using an OWL ontology. Instead of describing the ontology structure, we discuss the grammar of AMSL in the grammar discussed in this section, where [] represents an optional element, * represent an element that can be repeated from *0* to *n* number of times, and terminals are put between double quotes. An alert management strategy is represented by a sextuple:

Alert → ([AGSName], AlertName, [LifeCycle], [AlertConfig], PatientID*, UserID*).

UserID and *PatientID* show the list of users and patients that use this strategy for this alert. *AlertConfig* and *LifeCycle* are discussed in the rest of this section. *AGSName* names the AGS that is the source of the generated alert.

LifeCycle

LifeCycle represents the lifecycle of alerts and is defined as follows:

- LifeCycle → ([CfounterConfig], [SeverityConfig], [ActivationExpression], [InactivationExpression]).

CounterConfig of a *LifeCycle* defines a counter that represents the number of times an alert has been generated for a patient.

- CounterConfig → ([ResetDuration], [DurationBeforeIncrease]).

Alerts that are too old are deemed to be irrelevant, and hence should not be considered in the counter of that alert. The logic is as follows: if the time when the alert was last generated is more than *ResetDuration*, the counter for that alert is set to zero. Likewise, alerts that are generated too close to the previous one may be irrelevant as well. To account for this situation, if the duration between the last time the counter was incremented and the generation of the current alert is less than *DurationBeforeIncrease*, the counter will not be increased. As we will describe later, this counter is used in the calculation of severity and evaluation of activation and inactivation conditions.

- SeverityConfig→ ([MinSeverity], [MaxSeverity], [NumOfSteps]).

Severity of an alert during its lifecycle may change based on its counter number. Severity changes from *MinSeverity* to *MaxSeverity* in *NumOfSteps* steps as the counter increases. For instance, if the severity configuration is (0,1,10), severity of alert will linearly increase from 0 to 1 in 10 steps.

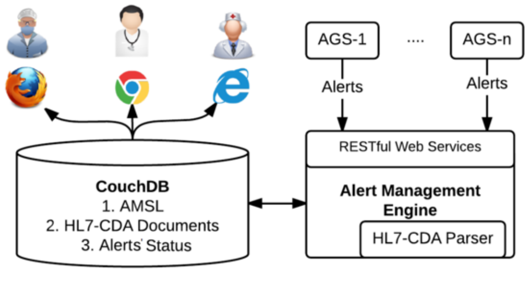

Figure 1 – Our Alert Management Framework

To avoid alert fatigue, an alert generated for a patient can be suppressed until the alert (or the patient) meets certain criteria. Until these criteria have not been met, the alert will not be activated (i.e. notified to the user). As an example, when a patient enters a high value for the blood pressure it will lead to generation of an alert. However, one high blood pressure value does not warrant immediate attention, as it may be a misreading or inaccurate measurement, hence such an alert can be suppressed, and only high blood pressure alerts that are generated within a specific time span may need medical attention. Another mechanism to avoid alert fatigue is to inactivate alerts that are not relevant anymore. An alert may be deemed irrelevant because its purpose has been accomplished. For instance, an active alert "Warfarin is recommended" can automatically become inactivated (i.e. hidden from user) when the patient is prescribed Warfarin. AME uses *ActivationExpression* and *InactivationExpression* in order to activate or inactivate an alert. An expression is defined as follows.

- Expression → Condition | (Expression | Condition, Operator, Expression | Condition).
- Operator → "or" | "and" .

Several types of conditions based on medications of a patient, lab test results, counter of the generated alert, or its severity may be defined:

- Condition → CounterCondition | MedicationCondition | SeverityCondition | LabResultCondition.

A *CounterCondition* is defined as follows:

- CounterCondition → (Comparator, Threshold).
- Comparator → "<" | ">" | "=" | "<=" | ">=".

Activation and inactivation of alerts may be based on their counter. For instance counter condition {"Comparator" : ">" , "Threshold" : 5} will be satisfied for a condition when its counter is 6 or greater.

- SeverityCondition → (Comparator, Threshold).

As we discussed previously, severity of a condition may change as the counter of the alert is changing. For instance, the severity condition (<, 0.5) is satisfied if severity of the generated alert is less than 0.5.

- MedicationCondition → (MedStatus | DosageChange | (Comparator, Threshold), MedicationName | MedicationCategory) .
- MedStatus → "active" | "onHold".

We can define conditions based on medications that exist in the health record of the patient. For instance, we can define a medication condition that is satisfied when a specific medication is *on hold*. Upon satisfaction of this condition based on patient record, the corresponding alert will be inactivated and removed from the list of active alerts shown to the user.

- LabTestCondition → ("existence" | (Comparator, threshold, unit) | TimesUpperLimit), labTestName.

Conditions based on lab test information of patients can also be defined. This type of condition can check existence of a specific lab test and whether its result is less or more than a specific threshold. The *TimesUpperLimit* value shows how many times value of the lab test result should be greater than the upper normal limit so that the corresponding condition is satisfied. For instance, {"existence", "labTestName" : "TSH"} will be satisfied if a TSH test result is present in the electronic medical record of the patient.

AlertConfig

AlertConfig represents how alerts are communicated with the user and aggregated in our framework. Receiving several alerts in one communication can potentially reduce the alert fatigue as relevant alerts are communicated and attended together.

- AlertConfig→ (CommunicationMedium, [AggregationConfig])
- CommunicationMedium → "SMS" | "e-mail" | "web-dashboard".
- AggregationConfig → ("non-aggregatable" | ("aggregatable", maxWait4Aggregation)).

CommunicationMedium represents the medium of communication for the generated alert. Alerts sent through a common communication medium can be aggregated if they are "aggregatable". *MaxWaitForAggregation* shows the maximum amount of time that sending a message can be delayed so that it can be aggregated with other messages. We will see an example of *AlertConfig* and alert aggregation in the next section.

Alert Management Engine (AME)

AME exposes RESTful web services that can be used by AGS to submit their generated alerts. Each incoming alert is a quadruple representing the AGS that has generated the alert, patient ID, the generated alert, and its time stamp. Incoming alerts will be used to update the corresponding JSON documents in CouchDB that represent the history and status of that alert-patient-user combination. In the rest of this section, we describe components and capabilities of AME.

Alert Management Algorithm

Upon receiving and incoming alert by AME, the following steps are performed in order to manage an alert:

1. Update alert's time stamps in CouchDB.
2. Update the alert's counter based on the time stamps.
3. Update alert's severity based on the counter.
4. Evaluate conditions.
5. Evaluate expressions based on conditions.
6. Alter state of alerts based on satisfaction of Inactivation and Activation Expressions.
7. Perform aggregation if possible.

HL7-CDA Parser

Medication and lab test conditions are evaluated based on patients' records. To connect to EMR, we assume patient records are received in Clinical Document Architecture (CDA) documents that is an XML based HL7 standard. HL7-CDA documents represent a snapshot of the patients' record at the time of creation. HL7-CDA documents are transformed to JSON and stored in CouchDB. HL7-CDA parser uses the Java JSON library to evaluate medication and lab test conditions. AME uses these evaluation results to calculate satisfaction of medication and lab test results conditions.

Adaptability and Personalization

We enable personalization in our framework by allowing each user-patient combination to have a separate configuration for each alert. Adaptability enables an alert management engine to tune its parameters in order to fit the needs of a specific user. Existing AGS provide this capability by enabling users to modify parts of the alert generation rules such as the threshold for the blood pressure alert rules. Tuning AGS rules can be time-consuming and requires extensive knowledge of the AGS domain. We propose a mechanism to provide adaptability based on user feedback. Users of AME can provide two types of feedback: (1) an alert is being activated too frequently for a patient, and (2) an alert is being activated less frequently than necessary. This feedback can be used to tune the alert lifecycle parameters accordingly in the following two ways:

(a) Based on the received feedback, parameters that are responsible for the activation frequency of the alert are multiplied by a pre-defined coefficient. For instance, if the feedback is that an alert is being activated less frequent than necessary, frequency parameters are modified in the following way: *resetDuration* *= 1.05, *DurationBeforeIncrease* *=0.95, *MinSeverity* *= 1/0.95, *steps* = round (1/0.95 * *step*). Hence, our alert management framework can adapt to users' needs based on the received feedback from them.

(b) If the user is knowledgeable about the internal workings of AME, he can choose a specific lifecycle parameter such as *ResetDuration* or *MinSeverity* to be modified based on his feedback. As a result, that specific lifecycle parameter is multiplied by a predefined coefficient. For instance, if the user indicates that an alert is generated more often than needed because of *ResetDuration* parameter, this value is multiplied by 0.8 so that the counter is reset more often leading to less frequent activation of that alert.

Aggregation

To understand how aggregation is performed, we go through a simple example. Suppose that two alerts are of the same *AlertConfig* as listed below:

```
{"AlertConfig": { "Aggregatable" : "yes" ,
   "AggregationMedium" : "e-mail",
   "maxWait4Aggregation" : "15m"}}
```

Imagine the following alerts are generated for two different patients (PatientIDs = 12 and 33) and are supposed to be sent to a specific physician (UserID = 7):

```
{"alert": "Warfarin is recommended",
 "patientID": 33, "UserID": 7, "Status" : "active"
 "DocumentID" : "1eb7ee96b33fa120c89"}
{"alert": "patient at high risk of bleeding",
 "patientID": 12, "UserID": 7, "Status" : "active"
 "DocumentID" : "9eb9ee96b88fa000c57"}
```

These two alerts are aggregated by creating the following document in CouchDB by AME:

```
{"DocumentType" : "aggregatedAlert"
 "AlertDocumentIDs" : ["9eb9ee96b88fa000c57",
                       "1eb7ee96b33fa120c89"]}
```

Alert Lifecycle Categories

In order to define the lifecycle of alerts in the IMPACT-AF project[1], we defined 3 general categories of alert lifecycles, made a template for each category, and fleshed out that template for each alert in that category. In the remainder of this section, we review these categories.

Category1: Cut Off Point Alteration

This category represents the lifecycle of alerts that compare patient data to a pre-defined cut off value. To avoid alert fatigue, these alerts can be ignored until that patient's value passes an alternative cut off value. As an example, according to the domain expert, output of the rule "*IF bilirubin is greater than its upper normal limit THEN patient might have abnormal liver function*" can be ignored until bilirubin is greater than two times the upper normal limit:

```
{"ActivationExpression" :
  {"LabTestCondition" :
      {" TimesUpperLimit" : 2,
       " MedicationName" : "bilirubin"}}
```

Bolded values in the above example can be replaced to represent lifecycle of other alerts in this category. In the same fashion, different alternative cut off values can be defined for dosage of medications

Category2: Medication and Lab Test based Lifecycles

By considering the type and dosage of medications taken by patients, several irrelevant alerts can be suppressed. As an example, according to the following rule, a heart rate below 60 bpm is considered abnormal: "*If (heart rate < 60) → patient's heart rate is abnormal*". However, this alert would be of less clinical significant if it is not associated with taking a rate control medication. Filtering low heart rate alerts that are not accompanied by such a medication will potentially avoid alert fatigue without compromising patients' safety or unnecessarily interrupting clinicians' workflow. We define the corresponding lifecycle as follows for this purpose:

```
{"ActivationExpression" :
  {"MedicationConditon" : {"MedStatus"  : "active" ,
        "MedicationCategory" : "rate control"}}}
```

As long as the above expression is not satisfied, alert *patient's heart rate is abnormal* will not be shown to the user. Bolded values in the above example can be replaced to represent lifecycle of other alerts in this category. As another example of this category, to avoid alert fatigue, alert pertaining to the following rule should be inactivated when patient starts taking warfarin: "*If the patient has CHA2DS2-VASc score >= 2 THEN alert physician that Warfarin is recommended*". To accommodate this alert fatigue prevention strategy, we define the lifecycle of the above rule as follows:

```
{ "InactivationExpression" :
  { "MedicationConditon" :
    {"MedStatus"  : "active" ,
       "MedicationName" : "Warfarin"}}}
```

Category3: Counter and Severity based Lifecycles

Some alerts can be suppressed until they are generated a certain number of times over a predefined period. For instance, Canadian guideline for management of Atrial Fibrillation recommend tailoring dose of heart rate control medications to bring resting heart rate of a patient to < 100 bpm. The corresponding rule is implemented as follows in the IMPACT-AF rule engine: "*If (heartrate > 100) → patient's heart rate is uncontrolled.*" Setting 100 as a cut point will probably fire too frequent alerts if patient is right on the edge. When presented this matter to the Cardiologist acting as the principal investigator in the IMPACT-AF project, he suggested ignoring this alert unless the average heart rate is more than 120 bpm over 72 hours with readings recorded at least 12 hours apart. If 24 hours passes and the rule engine does not generates any alerts, we start counting again. To accommodate this alert fatigue prevention logic, we defined the lifecycle of this alert as follows:

```
{"counterConfig" :  {"ResetDuration" : "24h" ,
  "DurationBeforeIncrease" : "12h"},
  "ActivationExpression" :  {"CounterCondition" :
"{"Comparator" : ">", "Threshold" : 5}}}
```

1 http://impact-af.ca/

Evaluation

To evaluate AMSL, five general practitioners were given a 20 minutes presentation on purpose of this language, its role in our framework, and several real world examples. Subsequently, participants were asked to assign an integer score from the range 1 to 5 (score 1 means that physician does not agree with the statement at all, and score 5 means that she completely agrees with the corresponding statement). Table 1 represents the percentage each score is chosen by the domain expert for the corresponding statement.

Table 1- AMSL Evaluation Statements and Percentages Each Score Is Give to each Statement[2]

Statements	Received Score				
	1	2	3	4	5
1.Purpose of AML elements are clear	0%	0%	0%	40%	60%
2. AML is easy to use	0%	0%	20%	60%	20%
3. AML can represent all aspects of alert fatigue prevention strategies	0%	40%	20%	20%	20%
4. All concepts in AML are useful for alert fatigue prevention	0%	20%	20%	40%	20%

Table 1 shows that majority of the participants find AMSL clear, easy to use, useful for and capable of representing alert fatigue prevention strategies. In the second part of the evaluation, we asked a domain expert to design 10 patient scenarios each composed of a sequence of 15 events making changes to health record of a patient (e.g., addition of a blood work or heart rate). Feeding these events to the IMPACT-AF rule engine generated 63 alerts. AME suppressed 34 of these messages. The remaining 29 alerts were aggregated into 14 atomic and composite alerts. We asked 5 general physicians to assign a number from the range 1 to 5 to statements 1, 2 and 3 for each aggregated alert, suppressed alert, and inactivated alert respectively. Table 2 represents the percentage that each score is given to the corresponding statement by domain experts.

Table 2 – AME Evaluation Statements and Percentages Each Score Is Give to each Statement[2]

Statements	Received Score				
	1	2	3	4	5
1. Aggregation is correct	3%	3%	11%	20%	63%
2. Suppression is correct	0%	1%	7%	21%	71%
3. Inactivation is correct	0%	0%	6%	4%	90%

Table 2 shows that most of the aggregations, suppression, and inactivation of messages have been performed successfully by AME according to the participants.

Conclusion

In this paper, we proposed a novel alert management framework that introduces unique features, such as: (1) a counter that filters effects of alerts that are too old or too new to be relevant, (2) alert severities that change by time, (3) possibility of defining conditions related to patients medications and lab test results, (4) adaptability based on users' feedback, (5) centralized management of alerts in case several AGS are generating alerts concurrently, (6) aggregation of alerts (possibly from different AGS) to reduce the number of alerts shown to the user, and (7) capturing communication medium information.

Moreover, our review of the literature shows that most of the alert management strategies are implemented for rule based decision support systems. Since our alert management framework is AGS dependent and does not make any assumptions regarding alert generation mechanism, it can be used to avoid alert fatigue in a variety of clinical decision support systems that use alternative technologies, such as neural networks or Bayesian belief networks.

An interesting future work is to explore the effects of alert management in general, adaptation, and personalization of alerts, in particular on patients' safety. For instance, in cases where physicians are heavily reliant on specific alerts for patients' safety, unpredictability of alerts due to varying parameters may potentially have perilous effects. In our framework, we defined a minimum severity to avoid this scenario. Other potentially dangerous scenarios should be identified and taken into consideration.

Acknowledgement

This research is supported by a research grant from Bayer Healthcare for the IMPACT-AF project

References

[1] McCoy AB, Thomas EJ, Krousel-Wood M, and Sittig DF. Clinical Decision Support Alert Appropriateness: A Review and Proposal for Improvement. Ochsner J . Ochsner Clinic Foundation 2014: 14(2): 195–202.

[2] Dolin RH, Alschuler L, Boyer S, Beebe C, Behlen FM, Biron P V, et al. HL7 clinical document architecture, release 2. J Am Med Informatics Assoc 2006; 13(1): 30–9.

[3] Graham KC, and Cvach M. Monitor alarm fatigue: standardizing use of physiological monitors and decreasing nuisance alarms. Am J Crit Care 2010: 19(1): 28–34.

[4] Van der Sijs H, Aarts J, Vulto A, and Berg M. Overriding of drug safety alerts in computerized physician order entry. J Am Med Inform Assoc 2006: 13(2): 138–47.

[5] Imhoff M, Kuhls S, Gather U, and Fried R. Smart alarms from medical devices in the OR and ICU. Best Pract Res Clin Anaesthesiol 2009: 23(1): 39–50.

[6] Klimov D, and Shahar Y. iALARM: An Intelligent Alert Language for Activation, Response, and Monitoring of Medical Alerts. In: Riaño D, Lenz R, Miksch S, Peleg M, Reichert M, ten Teije A, eds. Process Support and Knowledge Representation in Health Care SE - 10. Springer International Publishing, 2013; pp. 128–42.

Address for correspondence

Syed Sibte Raza Abidi, NICHE Research Group, Faculty of Computer Science, Dalhousie University, E-mail: ssrabidi@dal.ca

2 Score 1 means that physician does not agree with the corresponding statement at all and score 5 means that he completely agrees with it