2016 IEEE/ACM 38th IEEE International Conference on Software Engineering Companion

Use Runtime Verification to Improve the Quality of Medical Care Practice

Yu Jiang[1,2,3], Han Liu[3], Hui Kong[4], Rui Wang[1], Mohammad Hosseini[2], Jiaguang Sun[3], Lui Sha[2]

College of Information Engineering, Capital Normal University, China[1]
Department of Computer Science, University of Illinois at Urbana-Champaign, USA[2]
School of Software, Tsinghua University, China[3]
Institute of Science and Technology, Austria[4]

ABSTRACT

Clinical guidelines and decision support systems (DSS) play an important role in daily practices of medicine. Many text-based guidelines have been encoded for work-flow simulation of DSS to automate health care. During the collaboration with Carle hospital to develop a DSS, we identify that, for some complex and life-critical diseases, it is highly desirable to automatically rigorously verify some complex temporal properties in guidelines, which brings new challenges to current simulation based DSS with limited support of automatical formal verification and real-time data analysis.

In this paper, we conduct the first study on applying runtime verification to cooperate with current DSS based on real-time data. Within the proposed technique, a user-friendly domain specific language, named *DRTV*, is designed to specify vital real-time data sampled by medical devices and temporal properties originated from clinical guidelines. Some interfaces are developed for data acquisition and communication. Then, for medical practice scenarios described in *DRTV* model, we will automatically generate event sequences and runtime property verifier automata. If a temporal property violates, real-time warnings will be produced by the formal verifier and passed to medical DSS.

We have used *DRTV* to specify different kinds of medical care scenarios, and applied the proposed technique to assist existing DSS. As presented in experiment results, in terms of warning detection, it outperforms the only use of DSS or human inspection, and improves the quality of clinical health care of hospital.

Categories and Subject Descriptors

D.2.2 [**Software Tools and Engineering**]: Design Techniques; J.3 [**Life and Medical Science**]: Health

General Terms

computer-aided software engineering

ICSE'16 Companion, May 14-22, 2016, Austin, TX, USA
 ISBN 978-1-4503-4205-6/16/05 ...$15.00
DOI: http://dx.doi.org/10.1145/2889160.2889233

Keywords

Clinical Guideline, Real-time Data, Runtime Verification, Medical Decision Support System, Health Care.

1. INTRODUCTION

Within the scope of cyber-physical-human medical system, clinical guidelines and decision support sub-systems play a very important role in coordinating medical staffs to improve patient care. With interpreting text-based guidelines into some computer executable format, decision support systems are designed to monitor actions and observations during practice work-flow, and generate reminders and advice when corresponding guideline is not satisfied. Lots of evidence have showed that, using clinical guideline based decision support system, quality of medical care sometimes even the survival rate of patients, would be improved, and the medical care practice variability would be reduced [28]. However, with rapid developments of medicine science and computer technology, pathological model of some disease are becoming more and more precise and complex, and more real-time vital signs about patient need to be sampled and analyzed to prevent some complex non-deterministic temporal potential complications, which brings new challenges to current simulation based DSS.

For example, during the best practice guideline of ischemic stroke therapy [15], patient's neurological symptoms like speech difficulty and vital signs such as blood coagulation index should be monitored in real time, after the administration of recommended tissue plasminogen activator (rt-PA). If any of vital signs are out of range, stroke team will issue corresponding treatment orders to head nurse in ambulance or ICU to prevent patient from life-threatening complications such as hemorrhagic bleeding. Timely response based on real-time data monitoring and run-time rigorous verification is highly desirable, because a huge number of brain cells die every second. Another example is about the best practice guideline of infants respiratory distress syndrome, vital signs about blood-gas values should be monitored. If it is continuously above the normal range for at least 3 hours, the treatment is fine. But if it is too steep for at least 30 seconds, further emergency actions should be taken [5]. In both cases, specifying complex temporal properties for automatically rigorous verification would be more reliable than semi-automated manual vision inspection of current simulation based DSS.

More specifically, through the discussion with physicians, we learned that those phenomenons bring new challenges to

medical DSS in two aspects. First, although some guideline based DSS support static properties, some simple and rough temporal properties and clinical information sub-systems (ie. patient record systems) by simulation based precondition examination, ruinously run-time verification of complex temporal properties based on real-time data is not supported. Besides, not all vital signs will be stored in current patient record systems. Second, it not easy for medical staffs to monitor a huge number of vital signs for a long time, and input those conditions and violations into DSS in time. Statistics shows that medical staffs are under tremendous pressure and overloaded by a great amount of unorganized information. It is not easy to relate the information contained in the complex medical systems with temporal properties of complex complications with semi-automated manual vision inspection.

In practice, we propose a real-time data based runtime verification technique to cooperate with current DSS, to release medical staffs from safety critical complex temporal properties and huge amounts of vital signs. It combines formal methods in software engineering and practice guidelines in medicine to rigorously verify runtime temporal properties automatically, and can be implemented and plugged in existing DSS to strength health care. A domain specific language *DRTV* is proposed to specify the medical care scenario, as well as data and properties contained in this scenario. The proposed language inherits some features from existing computer interpretable formations of guideline for compatibility, while focuses more on data part and enhances property specification ability with past time linear temporal logic [16, 2]. Based on *DRTV* model, we develop a tool to automatically generate runtime monitors, which will continually extract scenario related data, parse their values to get event sequences, and input the event sequences to a runtime verification engine to rigorously check the temporal properties, accompanied with some interfaces for data communication. Overall, main contributions of our work are:

1. A runtime verification technique based on real-time data of patient and runtime verification is adapted to cooperate with medical decision support systems.

2. A domain specific language for specifying real-time data and complex temporal properties within a clinical scenario of best practice guideline are designed, and corresponding tools are implemented.

3. A real device based testing and evaluation are conducted to test the efficiency. To the best of our knowledge, this is the first study on applying runtime verification to improve the clinical health care.

The paper is organized as follows: related work is presented in Section 2. Some backgrounds on runtime verification and past time linear temporal logic are introduced in Section 3. Proposed run-time verification technique is presented in Section 4, including domain specific language, and run-time analyzer and verifier. Experiment results on some artificial examples and real applications plugged in real medical device are given in Section 5, learned lessons are presented in Section 6, and we conclude in Section 7.

2. RELATED WORK

Last decades, health care organizations and providers pay many efforts to guideline development and guideline based DSS implementation for daily health care. For guideline development, that is, structuring documents suggesting detailed steps to assist practitioner decisions about appropriate health care. Researchers pay attention to logic correctness and consistency to patient management technique and drug developments of the clinical guideline itself, and lots of formal methods have been adopted in verification and analysis of guidelines. Interactive theorem proving and model checking have been successfully applied to prove the consistency correctness of guideline properties. Some temporal quality properties of guideline have been captured based on the abductive diagnosis theory, further with the use of automated reasoning tools, such as the interactive theorem prover KIV or the use of program verification techniques [11]. Most of the analyzed properties are internal coherence properties of guideline and domain specific knowledge, while we conduct our work on the basis of well verified guidelines, mainly focus on the application of those guidelines.

For guideline application, that is, implementing well validated and verified practice guidelines into computer-based decision support systems to provide better health care. Actually, according to the Institute of Medicine, these systems improve the acceptance of guideline with automation of medicine practice. Text-based guidelines are represented and encoded into a computer-interpretable format, such as Arden [12], GLIF[26] , PROforma [8] and Asbru [5]. With the syntax and semantics of them, inference and decision making methodologies used in artificial intelligence such as rule-based reasoning, and probabilistic network can be designed and implemented. Then, decision support systems such as Spock [30] and CREDO [9] are developed to monitor actions and observations of medicine staff and provide corresponding suggestions, through task execution, condition examination, and procedure visualization. However, many clinical problems are complicated and involving many timely decision-making according to a huge number of real time vital signs. Then, task based simulation and staff observation based semi-automated collaboration of decision support systems encounter major difficulties. Rather than semi-automated informal methods such as artificial intelligence algorithms, our work will use runtime verification engine to deal with real time data automatically. In this way, real time rigorously verified results will be produced to assist current decision support systems.

For runtime verification, that is, verifying properties with runtime information of systems. It is effective to verify real time temporal properties, and has been applied in many applications. Within last ten years, considerable amount of work has been invested in program runtime verification systems [4]. These pieces of work are often extensions of AspectJ [13] for java programs. Some exceptions are ARACHE [7] and RMOR [10]. ARACHE performs runtime weaving into binary code of C programs with a limited form of regular expressions, while RMOR monitors the execution of C programs against state machines using aspect-oriented pointcut language to connect events to code fragments. For hardware runtime verification, property specification is usually translated into a hardware description such as VHDL and Verilog, which is then synthesized into a netlist and loaded into dynamically reconfigurable blocks of FPGA [27, 21]. Some work about the run-time verification of medical systems is

presented in [18, 19, 14]. They focus on mitigating safety hazards of closed-loop or open-loop medical systems [1, 3]. They work on runtime safety and reliability status of the system devices, hardware and communications, and nothing is related to decision support systems and clinical guidelines. For example, King et al. proposed a formal specification language to express and reason safety properties of on-demand medical systems [14]. Pajic et al. combined simulation-based analysis and model checking to guarantee the safety of closed-loop medical systems [25]. In [24], a model-driven approach allows us to prove safety properties of devices on the modeling level and ensures that the abstract models used in the verification process are sound with respect to actual dynamics of system. We conduct our work without considering the status of hardware systems, mainly focus on the real-time data of patient sampled in devices and their verification on temporal properties derived from clinical practice guidelines.

3. BACKGROUND

Runtime verification is the discipline of computer science that deals with the study, development, and application of those verification techniques that allow checking whether a run of a system under scrutiny satisfies or violates a given correctness property [17]. We use **past time linear temporal logic (ptLTL)** to specify the property, which provides temporal operators that refer to past states of an execution trace relative to a current point of reference [16]. The syntax and semantics are given as below.

ptLTL syntax: Let $AP = \{p_1, p_2 \cdots p_i \cdots p_n\}$ be a set of atomic propositions, then $ptLTL$ formulae is:

$$\phi, \psi ::= p_i \mid \neg\phi \mid X\phi \mid X^{-1}\phi \mid \phi \wedge \psi \mid \phi S \psi \mid \phi U \psi$$

Where U, S, X, X^{-1} stands for *"until", "since", "next"* and *"previous"* temporal operators respectively. Based on these basic operators and some standard abstractions, temporal operators such as *"eventually", "always", "always in the past"*, and *"eventually in the past"* can be defined and denoted as F, F^{-1}, G, and G^{-1}, respectively. From definition, we can see that $ptLTL$ extends classical logic LTL with the past modalities, which is good for complex temporal properties presented in clinical guideline.

ptLTL semantics: *Let ω be an infinite sequence $\omega = \omega_1\omega_2\cdots$, with a mapping $\eta : \forall i, \omega_i \rightarrow 2^{AP}$ labeling atomic propositions that hold in each position ω_i.* With the structure path (ω, η), a nonnegative integer i, and $ptLTL$ formulae ϕ and ψ, the relation that "ϕ holds at position ω_i" denoted as $\omega, i \models \phi$ can be inductively defined as below:

$$\begin{array}{lll}
\omega, i \models p & iff & p \in \eta(\omega_i) \\
\omega, i \models \neg\phi & iff & \omega, i \not\models \phi \\
\omega, i \models X\phi & iff & \omega, i+1 \models \phi \\
\omega, i \models X^{-1}\phi & iff & \omega, i-1 \models \phi \\
\omega, i \models \phi \wedge \psi & iff & \omega, i \models \phi \ \ and \ \ \omega, i \models \psi \\
\omega, i \models \psi S \phi & iff & \exists m \in [0, i], \omega, m \models \phi, \\
 & & and \ \ \forall n \in [j+1, i]\omega, n \models \psi \\
\omega, i \models \psi U \phi & iff & \exists m \in [i, \infty), \omega, m \models \phi, \\
 & & and \ \ \forall n \in [i, m)\omega, n \models \psi
\end{array}$$

Where two $ptLTL$ formulae ϕ and ψ are said to be equivalent, when condition "$\omega, i \models \phi \ \ iff \ \ \omega, i \models \psi$" is satisfied for all structure path ω and i. The equivalence relation between them can be denoted as $\phi \equiv \psi$.

4. VERIFICATION APPROACH

In this section, we introduce how the work-flow of real-time data based runtime verification technique cooperates with existing medical care systems, including domain specific language $DRTV$ used to specify data and properties of medical care scenario, and semantics formalization to formalize the scenarios described in a $DRTV$ model into the input sequence and automata for runtime verification.

4.1 Verification Work-flow Overview

The proposed real-time data based runtime verification technique is presented in Figure 1. First, we build a $DRTV$ model to specify vital real-time data sampled by medical devices or from Electronic Patient Record (EPR), and temporal properties originated from clinical guidelines. For data part specification, information such as data format, location, and type should be captured. For specification of property, $ptLTL$ formulae should be written correctly according to description of clinical guidelines. Also, event mapping expressions would be defined on current data, or historical data. Then, for medical practice scenario described in $DRTV$ model, we will formalized it for automatical generation of corresponding runtime property monitor with a developed engine based on MOP technique [22]. The automatically generated monitor will continually read real time data or historical data, verify temporal properties on them, and produce output to assist decision support systems to produce better health care.

Then, let us see the structure of runtime monitor. As presented in Figure 2, work-flow of the generated runtime monitor by our developed engine is based on three components: data parser, event path generator, and property verifier. These three components are automatically derived from $DRTV$ model with some formalization rules implemented in the engine. They will cooperate together to accomplish the task that real time patient data is processed to get runtime verification result, with main steps listed below:

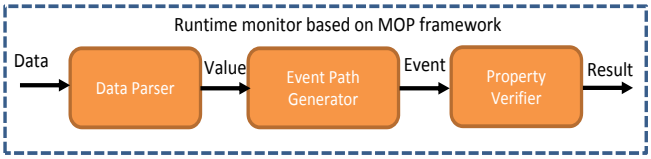

Figure 2: The runtime monitor generated according to the *DRTV* model.

1. **Data Parser:** This component is formalized and automatically generated according to the data description part of $DRTV$ model. With data parser, all relevant vital signs within model description will be abstracted from data packet sampled by medical devices, or from electronic patient records.

2. **Event Path Generator:** This component is formalized and automatically generated according to the event description part of $DRTV$ model. With event parser generator, values of vital signs abstracted in previous step will be used to evaluate boolean formula to get corresponding events.

3. **Property Verifier:** This component is formalized and automatically generated according to the property description part of $DRTV$ model. With property

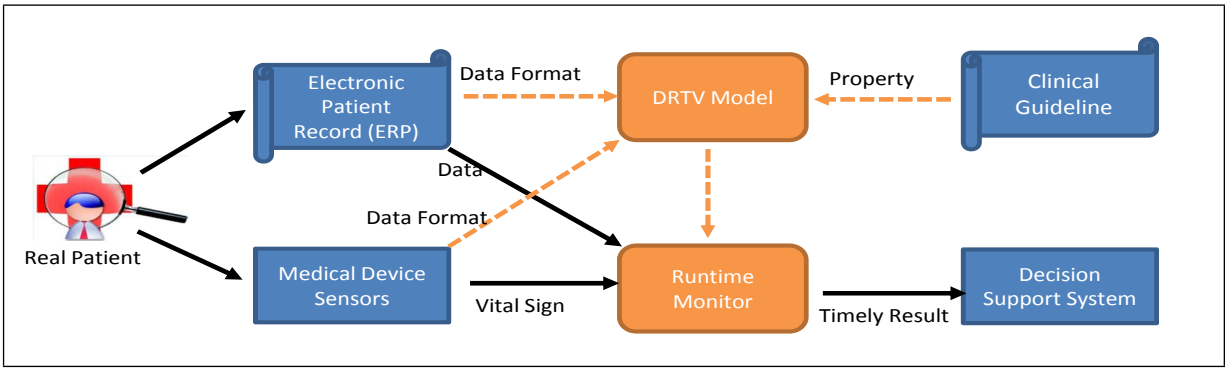

Figure 1: The real-time based run time verification technique, and interfaces with main components in current medical care systems .

verifier, events generated in previous step will be read to decide the transition of monitor automata. If the automata transit to a violation state, timely response should be produced to the decision support system to remind further actions of medical staffs. Otherwise, the monitor will continually read the event.

An example of monitor workflow is presented in Figure 3. Kernel of the real time data based runtime verification technique is the *DRTV* model, which is independent of the format of existing computer interpretable guidelines. Besides, the automatically generated runtime monitor through a developed engine based on MOP is running independently from current decision support systems. So, the proposed technique is platform independent, and could be customized and plugged into many existing medical care systems.

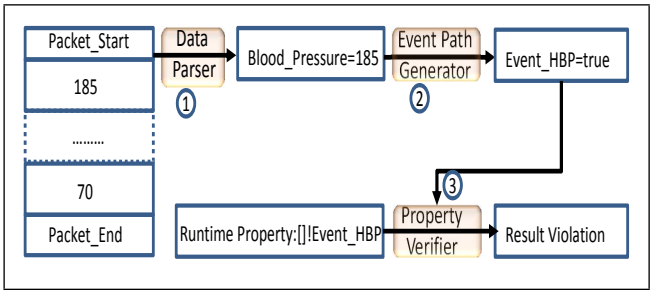

Figure 3: A runtime verification example to monitor blood pressure.

4.2 Domain Specific Language DRTV

The proposed domain specific language *DRTV* should provide the ability to describe data, event and temporal properties in different medical scenarios, based on which, we will formalize those elements to generate runtime monitor. The language should also be clear to use. We survey many formats of existing computer interpretable guidelines, and build our syntax on them, with more focus on data and *ptLTL* formula property specification. For example, the data specification part in *DRTV* makes use of some features from syntax of Arden, a widely used clinical guideline modeling language. In this way, medical staffs and engineers of medical systems will be more familiar to understand and construct a *DRTV* model, even when they have little experience in run-time modelling or verification. Kernel syntax of the domain specific language *DRTV* is presented as below.

Scenario module: Each *DRTV* model contains one module for a medical scenario. Each module starts with a reserved word *Scenario*, followed by scenario name and entity. The main entity consists of five constructs *data_resource, current_data, history_data, event,* and *property*. The first construct *data_resource* specifies resource of real time data. If the resource is medical device sensors, name and data packet length of the device need to be captured by construct *device_name* and *packet_length*. If the resource is electronic patient record, name and record length of the patient record need to be captured by construct *record_name* and *record_length*. The length will be used to help to locate and abstract data from data packet or patient record. Note that different medical device sensors and electronic patient record systems will use different kinds of information format for transmission.

$$
\begin{array}{lcl}
DRTV\_model & ::= & 'Scenario' < module\_name > \ < module\_entity > \\
module\_entity & ::= & < data\_resource > \\
 & & < current\_data > \\
 & & < history\_data > \\
 & & < event > \\
 & & < property > \\
data\_resource & ::= & 'Medical\ Device\ Sensors :' \\
 & & < device\_name >< packet\_length >; \\
 & & |'Electronic\ Patient\ Record :' \\
 & & < record\_name >< record\_length > \\
record\_length & ::= & integer \\
packet\_length & ::= & integer \\
device\_name & ::= & ('a'..'z'|'A'..'Z'|'\_')('a'..'z'|'A'..'Z'|'0'..'9'|'\_')* \\
record\_name & ::= & ('a'..'z'|'A'..'Z'|'\_')('a'..'z'|'A'..'Z'|'0'..'9'|'\_')* \\
module\_name & ::= & ('a'..'z'|'A'..'Z'|'\_')('a'..'z'|'A'..'Z'|'0'..'9'|'\_')*
\end{array}
$$

Vital sign: Current data and history data are captured in construct *current_data* and *history_data* contained in main entity, respectively. Because each packet or record may contain many vital signs corresponding to a set of indexes, the construct *current_data* is refined with a reserved word *current*, and a set of construct *index*. Each *index* specifies the type of value, position and length of vital sign contained in packet or record. Three basic data types (*Integer, String* and *Boolean*), and their corresponding array version (*Integer[], String[]* and *Boolean[]*), are supported. With the position and length of vital signs, and the length of data packet and record, value of vital signs will be located and abstracted exactly.

```
current_data  ::= 'current' : (< index >) *
index         ::= < data_type > < data_name > < data_location >
data_type     ::= 'int' | 'bool' | 'string' |
                  'int[]' | 'bool[]' | 'string[]'
data_location ::= < data_position > < data_length >
data_position ::= 'data_start_position :' integer
data_length   ::= 'data_length :' integer
```

The construct *history_data* is refined with a reserved word *history*, and a set of construct *history_index*. Each *history_index* specifies the type of value, initial value, and update rule of this history index. Supported data types for history data are the same with current data. The update rule is some general java statements that abstract the value, according to the previous packets and records. Also, the update rule can be constructed based on the position and length of *current_data*, with an additional integer to denote the number of packets needs to be searched ahead.

```
history_data  ::= 'history' : (< history_index >) *
history_index ::= < data_type > < data_name > < abstract_rule >
data_type     ::= 'int' | 'bool' | 'string' |
                  'int[]' | 'bool[]' | 'string[]'
abstract_rule ::= < ahead_num >< data_position >< data_length >
                  | java − assignment − statement
ahead_num     ::= 'previousnumber :' integer
data_position ::= 'data_start_position :' integer
data_length   ::= 'data_length :' integer
```

Real-time event: Event is defined in the fourth construct *event* contained in the main entity. Because vital signs contained in each packet or record may indicate several events, a set of identifiers is used to differentiate them. Traditionally, each event is defined as some boolean expressions on current data. But in case of some temporal properties, event may also use history data. Computations on these data are supported for events related to complex decision logic. We can use just one event with a boolean expression on these data, to release medical staffs from keeping monitoring device screen of multiple traces.

```
event      ::= 'event :' ( < event_name >'='< bool_exp >) *
bool_exp   ::= < current_data_name >
               | < history_data_name >
               | < bool_value >
               | < comput_exp >< compar_op >< comput_exp >
               | < bool_exp > '&' | '||' < bool_exp >
               | '!' < bool_exp >
               | '(' < bool_exp > ')' ;
comput_exp ::= < current_data_name >
               | < history_data_name >
               | < int_value >
               | < comput_exp >< arithe_op >< comput_exp >
               | '(' < comput_exp > ')' ;
arithe_op  ::= '/' | '%' | '+' | '−' | '*' ;
compar_op  ::= '==' | '!=' | '<' | '≥' | '≤' | '>' ;
```

Temporal property: Temporal property corresponding to events is defined in the fifth construct *property* of main entity. It consists of two parts, a set of *ptLTL* formulae and handlers. Both parts can be derived from clinical best practice guidelines, such as the logic description part of Arden. Some newly developed medicine knowledge that are not presented in current clinical guidelines can also be encoded into these two parts. The *ptLTL* formulae provide ability to describe most static and temporal conditions, and the handlers will take the result of verification to produce timely response to medical staffs. Besides the standard temporal operators, extra temporal operators such as *"eventually", "always", "always in the past",* and *"eventually in the past"* denoted as F, F^{-1}, G, and G^{-1}, are also explicitly encoded in the construct. If the property is violated, suggested actions and statements within the handler will be executed, and the runtime monitor would be reset to the initial state automatically. The warnings as well as some status information can be encoded in the handler construct through some print statements of standard Java language.

```
property   ::= 'property :'
               ( < propoety_name >'='
               < ptLTL_exp > < handler >) * ;
ptLTL_exp  ::= event_name
               | ¬ptLTL_exp
               | X ptLTL_exp | X^-1 ptLTL_exp
               | F ptLTL_exp | F^-1 ptLTL_exp
               | G ptLTL_exp | G^-1 ptLTL_exp
               | ptLTL_exp ∧ ptLTL_exp | ptLTL_exp S ptLTL_exp
               | ptLTL_exp U ptLTL_exp
handler    ::= java − statement
model_name ::= ('a'..'z'|'A'..'Z'|'_')('a'..'z'|'A'..'Z'|'0'..'9'|'_') * ;
```

Based on those syntax definitions presented above, we can build a scenario module within a *DRTV* model. Real-time vital signs and complex temporal restrictions on these signs of patient could be described in a structured manner. Then, the runtime monitor, consisting of the data parser, event path generator, and property verifier will be formalized and generated automatically from a developed engine, as described in the following subsection.

4.3 Semantics and Monitor Formalization

With scenarios described in the *DRTV* model, we need to define the semantics to formalize the data, event, event paths, and property verifier automata for computer interpretable formal verification as below.

Vital data formalization: For the current data, which are corresponding to the vital signs of patient, they are formalized as a variable set $\mathbb{D}$ derived from the construct *current_data*. The type of each data d derived from the construct *data_type* is denoted as $T(d)$, where $T(d) \in \{$*Integer, String, Boolean, Integer[], String[], Boolean[]*$\}$.

In the same way, we can formalize the history data. They are formalized as a variable set $\mathbb{D}^h$ derived from the construct *history_data*, and the type of each data d^h is the same with their corresponding current data d. For each variable $d \in \mathbb{D}$, there may be several $d_i^h \in \mathbb{D}^h$ used to capture different time nodes of history. For the history variable, extending its assignment with some general computations on history values is optional for complex restrictions. Assignment of the data set is dependent on the information contained in sampled packets and records, formalized as below,

> ***Formalization 1:*** *θ is a full assignment to $\mathbb{D}$ on the domain of type, where $\theta(d)$ is the value of data d contained in θ, which is derived from the data_location construct. θ^h is a full assignment to $\mathbb{D}^h$, where $\theta(d^h)$ is the value of data d^h contained in θ^h, which is derived from the abstract_rule construct.*

where θ is a new sampled data packet or a new electronic patient record item, and θ^h is a previously sampled packet or history record.

Real-time event formalization: After the data value assignment is abstracted from data packets or records, boolean expressions described in the construct *bool_exp* should be evaluated to get the event set initialized. All events indicated in a data packet or record need to be addressed correctly. Let $\mathbb{E}$ be a set of events derived from the construct *event_name*, and the data set related to the event set is denoted as $\mathbb{D}_{\mathbb{E}}$, where $\mathbb{D}_{\mathbb{E}} \in \mathbb{D} \cup \mathbb{D}^h$. Then, the event set is formalized as:

> ***Formalization 2:*** *$\forall e \in \mathbb{E}$, e is a full assignment to the boolean expressions on $\mathbb{D}_{\mathbb{E}}$. Event e is said to be happened when the assignment is evaluated to be true, which is denoted as $e(\theta(\mathbb{D}_{\mathbb{E}})) == true$.*

For event path, it is more complex, because each packet or record may indicate more than just one event corresponding to different indexes. The path should be defined as a sequence of set, where each set a_i is the combination of events evaluated to be true. It is a subset of all events contained in $\mathbb{E}$. Then, the event path is formalized as:

> ***Formalization 3:*** *π^* is the group of all finite set sequence π ($\pi = \pi_1\pi_2\pi_3\cdots\pi_n$), and π^ω is the group of all infinite set sequence $\pi^{'}$ ($\pi^{'} = \pi_1^{'}\pi_2^{'}\pi_3^{'}\cdots$). Each π_i contained in the set sequence is the event combination evaluated to be true in the data packet or record θ_i, denoted as $\pi_i = \{e_j | e_j(\theta_i(\mathbb{D}_{\mathbb{E}})) = true\}$ and obviously $\pi_i \in 2^{\mathbb{E}}$. Furthermore, if $\forall i \in [1, n], \pi_i = \pi_i^{'}$, then $\pi_i^{'}$ is an extension of π. All possible extensions of the finite path π are denoted as $\Sigma(\pi)$.*

Take the example presented in Figure 3 to demonstrate the event and path formalization and generation. There are two indexes conveyed in the sampled data packet, *blood pressure* and *heart rate*. We can define six events on these two data through different boolean expressions.

$$\begin{array}{lcl} Event\_HBP & := & (Blood\_Pressure > 180) \\ Event\_LBP & := & (Blood\_Pressure < 90 \\ Event\_NBP & := & (Blood\_Pressure \leq 180 \\ & & \wedge Blood\_Pressure \geq 90) \\ Event\_HHR & := & (Heart\_Rate > 120) \\ Event\_LHR & := & (Heart\_Rate < 60) \\ Event\_NHR & := & (Heart\_Rate \leq 120 \\ & & \wedge Heart\_Rate \geq 60) \end{array}$$

Based on basic events, some complex events can be defined as below. They can also be defined on those indexes directly with more complex boolean expressions.

$$\begin{array}{lcl} Event\_Safe & := & (Event\_NHR \wedge Event\_NBP) \\ Event\_UnSafe & := & (Event\_LHR \vee Event\_HHR \\ & & \vee Event\_LBP \vee Event\_HBP) \end{array}$$

Those events will be evaluated when there comes a data packet or a patient record. The event set contained in the data packet of Figure 3 is {*Event_HBP, Event_NHR, Event_Unsafe*}. A path for continually sampled packets of medical care system might be {*Event_HBP, Event_NHR, Event_Unsafe*}, {*Event_NBP, Event_HHR, Event_Unsafe*}, {*Event_NBP, Event_NHR, Event_safe*} $\cdots$.

Temporal property formalization: The property derived from the construct *property* is the verifier that partitions path into three types, *violation*, *validation* , and *unknown*. Then, the property verifier is formalized as:

> ***Formalization 4:*** *A property verifier derived from the ptLTL formula ϕ is a full assignment to π^* on the domain {violation, unknown, validation}, where $\forall \pi \in \pi^*$, the assignment rule is:*
>
> - *If $\forall\ \pi^{'} \in \Sigma(\pi), \pi^{'} \models \phi$, then $\phi(\pi) = validation$*
> - *If $\forall\ \pi^{'} \in \Sigma(\pi), \pi^{'} \not\models \phi$, then $\phi(\pi) = violation$*
> - *Else $\phi(\pi) = unknown$*

Condition of the assignment rule can be realized by the equivalent monitor automata of the *ptLTL* formula, which can be customized and automatically generated within MOP [22, 2]. The automaton formalized as below is used to monitor the event sequences defined on the real-time vital signs of patient, and the condition $\pi^{'} \models \phi$ is satisfied when the corresponding path is accepted by the automaton.

> ***Formalization 4:*** *The customized monitor automaton corresponding to the plLTL formula ϕ is defined as a tuple $\langle S, s_0, \alpha, L, O\rangle$, where:*
>
> - *$S = \{s_0, \cdots, s_n\}$ is the set of states*
> - *s_0 is the initial state*
> - *$\alpha = \{\alpha_0, \cdots, \alpha_n\}$ is the set of events contained in the formula ϕ, and $\alpha_i \in 2^{\mathbb{E}}$*
> - *$L = \{l_0, \cdots, l_n\}$ is the transition, and $l_i \in S \cdot \mathbb{E} \cdot S$*
> - *$O = \{o_0, \cdots, o_n\}$ is the output that maps the state to {violation, validation, and unknown}.*

Take the scenario presented in Figure 3 as an example. If there is a requirement that patient is not allowed to exceed the safe threshold of blood pressure, which can be defined on events *Event_HBP* and *Event_LBP*. This requirement can be formalized as a *ptLTL* formula presented below:

$$[\ ](notEvent\_HBP \wedge notEvent\_LBP)$$

Then, the customized and generated monitor automaton corresponding to this formula is depicted in Figure 4.

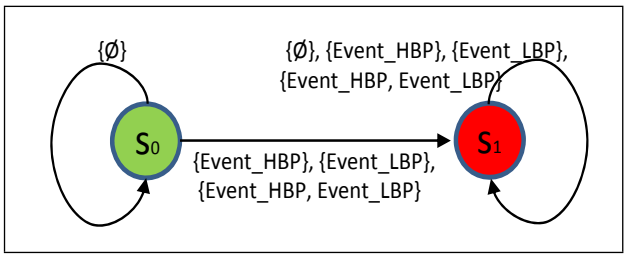

Figure 4: The formalized monitor automaton.

The automaton will start in the initial state s_0. When any of the event sets labeled on the transition happens, such as {*Event_HBP*} or {*Event_LBP*}, or both of them {*Event_HBP, Event_LBP*}, the automaton will transit to violation state s_1. If there is no event, the automaton will stay in the initial state s_0. The automaton only focuses on the event that is related to the property, while others will not be considered for efficient path classification.

Tool Implementation: Based on above syntax and formalization semantics, we implement an interface for *DRTV* model construction and an engine to automatically translate the *DRTV* model into the executable runtime monitor, which consists of data abstractor, event sequence generator, and property verifier. The interface also contains a backend to help validate the syntax correctness and store the model in .XML format. Then, the translator contained in the engine will parse the XML file to executable java files. In order to get real time data from the medical devices or electronic patient recode systems, we also develop a communication interface Sink for data transfer [23]. The overall structure of the tool implementation and interface cooperation is presented in Figure 5.

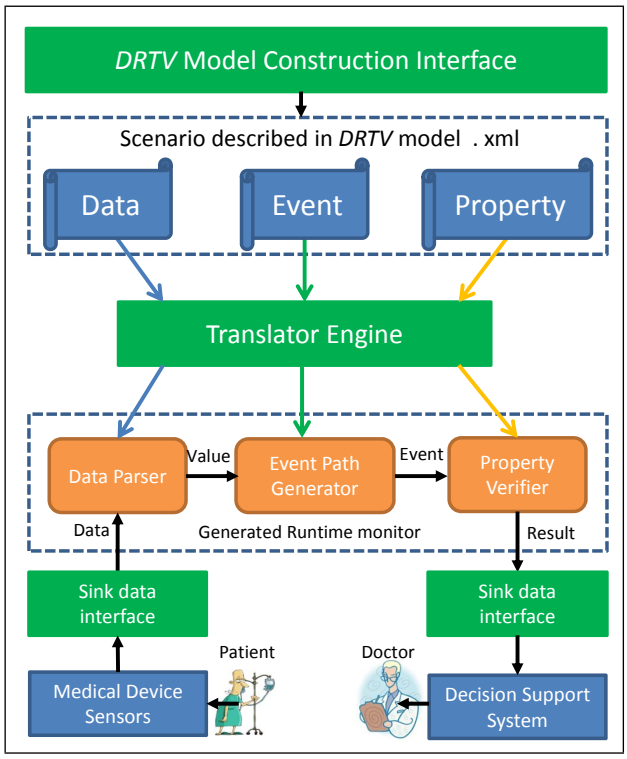

Figure 5: Tool implementation and interaction. The green modules are implemented components, the yellow modules are generated automatically, the blue files are input by the domain model engineers, and the blue modules are existing decision support systems or medical device sensors.

5. EXPERIMENT RESULTS

In order to evaluate the efficiency and scalability of the proposed real-time data based runtime verification technique, we apply it to the best practice guidelines of real medical care scenarios, then accomplish some real medical device based simulation with the closed collaboration of Carle Foundation Hospital. We conduct experiments and generate different runtime verifiers with consistency to a previous developed decision support system [1] which contains integrated workflow, data to decision pipeline, and Medical Device Plug and Play (MDPnP).

The first scenario for test is a best practice guideline recommendation for stroke care [20]. According to the guideline, for the ischemic stroke patient who meets proper criteria [15], administration of IV rt-pa is recommended in a dose of 0.9 mg/kg (maximum of 90 mg), with 10% of the total dose given as an initial bolus and the remainder infused over 60 minutes. Since a major risk for patient using IV rt-PA is the complication of brain hemorrhage, patient's neurological symptoms such as speech difficulty, facial droop, weakness in hands and vital signs such as the blood pressure, heart rate, SpO2 and blood glucose level index will be monitored in real time. If any of the vital signs are out of range, stroke team will issue corresponding treatment orders.

```
Scenario : Stroke_Care
  Medical Device Sensors : intellivue mp70  Data Length : 64;
    Current :
        Int Blood_Pressure , Data Start : 8 , Data Length : 8;
        Int Heart_Rate, Data Start : 8 , Data Length : 8;
    Event :
        Event_UNBP  = (Blood_Pressure > 180  || Blood_Pressure < 90 );
        Event_UNHR  = (Heart_Rate > 120  &&  Heart_Rate < 60 );
    Property:
        SafeState = [](not (Event_UNBP \/ Event_UNHR );
        Handler_SafeState{
            Print("Blood :"+Blood_Pressure+"Heart:"+ Heart_Rate);
            Print("Warnings!!!" + ring());
        };
End Scenario
```

Figure 6: *DRTV* model for the blood pressure and heart rate runtime verification, where EVENT_UNBP and EVENT_UNHR denotes the un-normal events leading to property violation.

For example, when the patient's blood pressure exceeds the safe threshold 180, the stroke team may suggest injecting nitroprusside to control the blood pressure. If the nitroprusside infusion causes the neural deterioration, the physician may change the drug accordingly. If blood pressure and blood glucose level cannot be controlled under acceptable ranges, or signs of brain hemorrhage appear, the stroke team may stop the rt-PA and adjust its schedule to treat complications. Timely response based on the real-time data monitoring and runtime rigorous verification is highly desirable, because every second the huge number of brain cells die, for example, 32000 brain cells will die within every second a clot blocks blood flow to brain. It is not easy for staff to keep the neurological testing and vital sign monitoring for a long time, but a simple *DRTV* model segment for blood pressure and heart rate runtime verification can be modeled, as presented in Figure 6.

Other data parameters such as SpO_2 and temperature, and some corresponding events corresponds to this scenario can also be declared in the model. Those kind of static properties can also be supported by existing decision support systems such as Spock and CREDO. But for some temporal properties such as when an event indicates the high blood pressure, the following event must indicate the nitroprusside injection, it can also be defined as

$$[\,](Event\_HBP \;\; X \;\; Event\_Nitroprusside)$$

which is not supported in Spock and CREDO. Then, the generated runtime verifier will be used to verify the real

[1]The system and related video is presented in `http://publish.illinois.edu/mdpnp-architecture/advanced-situation-awareness/`

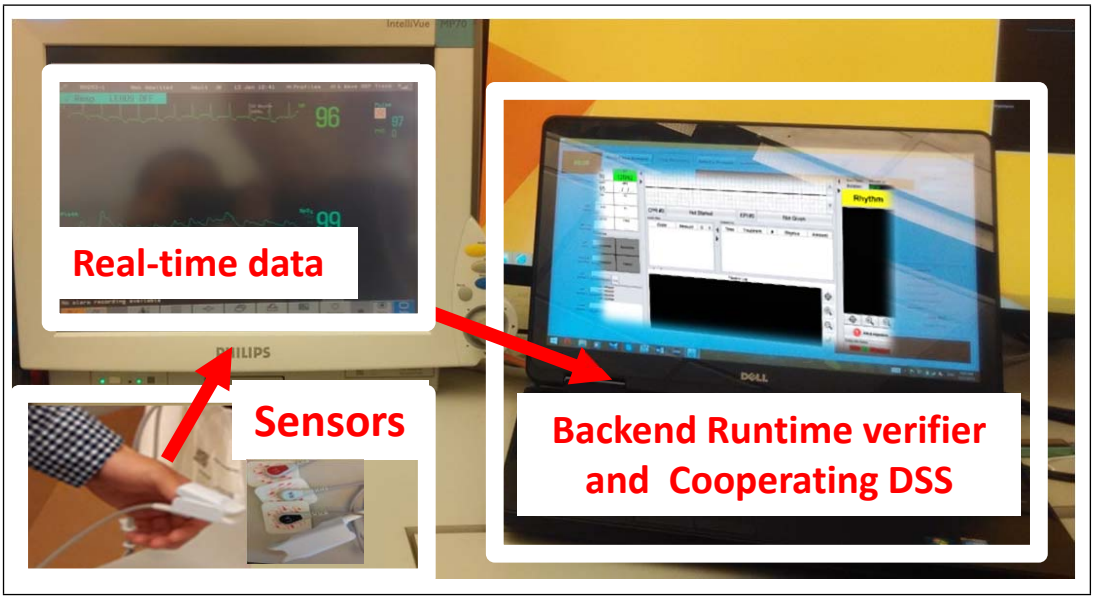

Figure 7: The real data and device based runtime simulation and verification.

time data, and produce the timely response automatically, as presented in Figure 7. The real time data monitor device Phillips IntelliVue MP70 with the sensors attached on it are used to sensor the real-time data, and the derived runtime verifier of the *DRTV* model and the data interface are running on the computer. With the developed data interface and MDPnP driver, we continually get the data from Phillips IntelliVue Mp70 and pass them to the generated runtime verifier or our implemented decision support system directly. Initially, the real-time data sampled from myself will not violate the property described in Figure 6. It is not easy to adjust my blood pressure 118 to trigger the violation of property, so inverting the property for testing is adopted in lab simulation. When the property verified is $[\,]\ Event\_UNP$, timely warnings is produced immediately.

We also do tests on some more complex guidelines, such as the guideline of infants respiratory distress syndrome. If the blood-gas values are too steep for at least 30 seconds, warnings and further actions should be taken. It is not easy to decide the condition continuously steep of 30 seconds with semi-automated manual vision inspection. But with the proposed lightweight runtime verification technique, it can be automatically monitored by the following *ptLTL* formula encoded in *DRTV* model.

$$[\,](\underbrace{Event\_Steep\_BG \quad X \quad \cdots X \quad Event\_Steep\_BG}_{})$$

Where $Event\_Steep\_BG$ is defined as a boolean expression on two consecutive data packets ($BloodGlas - BloodGlas_1^h \geq Steep\_Threshold$), one for current data packet and one for the history data packet. When there are 30 number of consecutive steep blood-gas incensement, the property would be violated, and timely warning will be responded immediately.

After those tests, we use SimMan patient simulator to set the vital signs of virtual patient and the value of real time data monitor device Phillips IntelliVue MP70, which can be furthered passed to the cooperating decision support system and generated runtime verifier. In this way, more guideline properties for potential complications can be verified with different kinds of values set by the SimMan patient simulator, and the results come as expected.

Table 1: Detected warning comparisons for different scenarios, and the symbol ∅ means not support.

Property	Number	Manually	DRTV	Spock
UN_HBP	10	10	10	10
Steep_BG_20	10	10	10	∅
Steep_BG_30	10	9	10	∅
UN_HBP	100	94	99	100
Steep_BG_20	100	81	98	∅
Steep_BG_30	100	76	98	∅
UN_HBP	1000	931	992	991
Steep_BG_20	1000	773	993	∅
Steep_BG_30	1000	645	992	∅

Furthermore, we choose three typical properties to help us test the efficiency, with results presented in table I. The first column of table I is the property name which is defined in lab test above, the second column is the number of violations we insert into the virtual patient through SimMan patient simulator, the third column is the violations manually detected by staring at real-time data monitor device, the fourth column is the violations detected with the accompanied runtime verifier, and the fifth column is the violations detected with the accompanied Spock DSS. From the trend of the third column, we can find that the accuracy decreases along with the complexity of the property and the work time. For the runtime verifier, it performs steadily as in the fourth column. Noting that the DRTV and Spock will produce 10, 100 or 1000 number of warnings, but one or two percent may also be ignored because of noise or other effects that disturb them. According to the simulation, it is reasonable to draw the conclusion that the lightweight runtime verifier cooperating with the existing DSS running on the computer helps produce an easier health care practice.

6. LESSONS LEARNED

(a) Physicians need more flexible and automatic support techniques to release them from the huge number of human tasks during the clinical health care: Nowadays, along with the development of medicine science, more and more medical devices are placed in the ward to provide the information to assist in making decision. However, these devices provide an extra dimension of information for physicians to process [6]. Physician can miss read, miss interpret, mixed use the provided information or recall incorrect knowledge to make a decision, during the increasingly common case of continues long time stressing work.

For example, the work [29] shows that, although medical staffs practice the best practice for cardiac arrest resuscitation, due to its urgent and infrequency on a daily basis, medical staffs may be panic at the situation and miss several warnings. Our experiments also support the conclusion of the above work, and further show that current DSS does not perform that much good when coming to complex properties, and recently developed computer technology needs to be incorporated. We make use of runtime verification technique, to release physicians from tremendous pressure to relate the information contained in complex medical care systems with temporal properties by semi-automated manual vision inspection of current DSS.

(b) Easy to use interfaces are needed to facilitate physicians to use formal verification: Computer technologies such as runtime verification and domain specification language are totally new to physicians, we need user-friendly interfaces to convince and facilitate physicians to believe and use those techniques. Currently, it is not possible for physicians to pay extra efforts to learn those techniques due to a huge amount of clinical works, we need to reduce their work by hiding details of implementation techniques and provide the least complex interface of scenario description language *DRTV*.

For example, during the design of *DRTV*, we plan to use syntax similar to java, which can be more easily connected to the back-end MOP. but the physicians from Carle Foundation Hospital thought that it is not easy for them to understand and build the model. Hence, we reduce their efforts by searching many description languages used in current medical DSS, and inherit some syntax from them with assigning MOP related semantics. Also, the physicians suggest that it would be better for us to provide some templates to translate the property described in the medical best practice guideline to the property described for in *DRTV* in our future work, which will facilitate their practice of our approach in their health care practices. If the scenario modeling process can be accomplished by automatical generation based on the configuration of medical best practice guideline and devices, it will be more fascistic for them.

(c) Actionable verification result is needed to facilitate physicians to use verification results: Additional notations and actions for explaining the verification results are needed to make verification results more actionable for physicians. In contrast to most ordinary applications that allow us to analysis the violations and figure out which parts should be responsible for its violation, violations in medical care practice usually need timely response of physicians. We have provided the handler construct for the property description of *DRTV*.

7. CONCLUSION

In this paper, we propose a lightweight real-time data based runtime verification technique for medical care practice. First, a user-friendly domain specific language *DRTV* for specifying the real-time data and complex temporal properties of the medical care practice is designed. Based on the *DRTV* model, a runtime verification technique is proposed and formalized to strengthen the medical decision support system. It combines formal methods in software engineering and practice guidelines in medicine to rigorous verify runtime temporal properties automatically, and can be implemented and plugged in existing medical support systems to strength the health care.

Discussion: (1) According to the discussions with doctors, the *ptLTL* property specification in *DRTV* model may be easy for computer science staffs, but it is not easy for medical staffs. It is better to provide some easier templates and improve the user-friendly of the language. (2) We need to develop more data interfaces to support more existing decision support systems and medical devices. Then, properties on more complex scenarios with data from multiple monitoring devices at the same time would be supported. (3) Right now, the runtime verification is mainly focused on those vital signs, we do not look into the pathophysiological model of patient. If we combine the runtime verification model with the organ automata of pathophysiology, more complete verification results might be produced.

Acknowledgement

The authors thank Dr. Bobby and Dr. Hill at Carle Hospital, Urbana, IL for their help with the discussion on medical knowledge. This work is supported by NSF CNS 13-30077, NSF CNS 13-29886, NSF CNS 15-45002, and NSFC 61303014.

8. REFERENCES

[1] D. Arney, M. Pajic, J. M. Goldman, I. Lee, R. Mangharam, and O. Sokolsky. Toward patient safety in closed-loop medical device systems. In *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*, pages 139–148. ACM, 2010.

[2] A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for ltl and tltl. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(4):14, 2011.

[3] L. Bu, Q. Wang, X. Chen, L. Wang, T. Zhang, J. Zhao, and X. Li. Toward online hybrid systems model checking of cyber-physical systems' time-bounded short-run behavior. *ACM SIGBED Review*, 8(2):7–10, 2011.

[4] F. Chen and G. Roşu. Java-mop: A monitoring oriented programming environment for java. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 546–550. Springer, 2005.

[5] P. A. De Clercq, J. A. Blom, H. H. Korsten, and A. Hasman. Approaches for creating computer-interpretable guidelines that facilitate decision support. *Artificial intelligence in medicine*, 31(1):1–27, 2004.

[6] Y. Donchin, D. Gopher, M. Olin, Y. Badihi, M. R. Biesky, C. L. Sprung, R. Pizov, and S. Cotev. A look

into the nature and causes of human errors in the intensive care unit. *Critical care medicine*, 23(2):294–300, 1995.

[7] R. Douence, T. Fritz, N. Loriant, J.-M. Menaud, M. Ségura-Devillechaise, and M. Südholt. An expressive aspect language for system applications with arachne. In *Transactions on Aspect-Oriented Software Development I*, pages 174–213. Springer, 2006.

[8] J. Fox, N. Johns, and A. Rahmanzadeh. Disseminating medical knowledge: the proforma approach. *Artificial intelligence in medicine*, 14(1):157–182, 1998.

[9] J. Fox, V. Patkar, and R. Thomson. Decision support for health care: the proforma evidence base. *Informatics in primary care*, 14(1):49–54, 2006.

[10] K. Havelund. *Runtime verification of C programs*. Springer, 2008.

[11] A. Hommersom, A. Hommersom, P. Groot, P. Groot, P. J. Lucas, M. Balser, and J. Schmitt. Verification of medical guidelines using background knowledge in task networks. *Knowledge and Data Engineering, IEEE Transactions on*, 19(6):832–846, 2007.

[12] G. Hripcsak, P. D. Clayton, T. A. Pryor, P. Haug, O. Wigertz, and J. Van der Lei. The arden syntax for medical logic modules. In *Proceedings/the... Annual Symposium on Computer Application [sic] in Medical Care. Symposium on Computer Applications in Medical Care*, pages 200–204. American Medical Informatics Association, 1990.

[13] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of aspectj. In *ECOOP 2001âĂŤObject-Oriented Programming*, pages 327–354. Springer, 2001.

[14] A. L. King, L. Feng, O. Sokolsky, and I. Lee. A modal specification approach for on-demand medical systems. In *Foundations of Health Information Engineering and Systems*, pages 199–216. Springer, 2014.

[15] M. G. e. Lansberg. Antithrombotic and thrombolytic therapy for ischemic stroke: antithrombotic therapy and prevention of thrombosis: American college of chest physicians evidence-based clinical practice guidelines. *CHEST Journal*, 141(2_suppl):e601S–e636S, 2012.

[16] F. Laroussinie, N. Markey, and P. Schnoebelen. Temporal logic with forgettable past. In *Logic in Computer Science, Symposium on*, pages 383–383. IEEE Computer Society, 2002.

[17] M. Leucker and C. Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

[18] T. Li, F. Tan, Q. Wang, L. Bu, J.-n. Cao, and X. Liu. From offline toward real-time: A hybrid systems model checking and cps co-design approach for medical device plug-and-play (mdpnp). In *Cyber-Physical Systems (ICCPS), 2012 IEEE/ACM Third International Conference on*, pages 13–22. IEEE, 2012.

[19] T. Li, F. Tan, Q. Wang, L. Bu, J.-N. Cao, and X. Liu. From offline toward real time: A hybrid systems model checking and cps codesign approach for medical device plug-and-play collaborations. *Parallel and Distributed Systems, IEEE Transactions on*, 25(3):642–652, 2014.

[20] P. Lindsay, M. Bayley, C. Hellings, M. Hill, E. Woodbury, S. Phillips, et al. Canadian best practice recommendations for stroke care (updated 2008). *Canadian Medical Association Journal*, 179(12):S1–S25, 2008.

[21] H. Lu and A. Forin. The design and implementation of p2v, an architecture for zero-overhead online verification of software programs. Technical report, Technical Report MSR-TR-2007–99, Microsoft Research, 2007.

[22] P. O. Meredith, D. Jin, D. Griffith, F. Chen, and G. Roşu. An overview of the mop runtime verification framework. *International Journal on Software Tools for Technology Transfer*, 14(3):249–289, 2012.

[23] H. Mohammad, Y. Jiang, W. Poliang, B. Richard, and S. Lui. Sink : A middleware for synchronization of heterogeneous software interfaces. In *ARM, 2015*, pages 1–6. ACM, 2015.

[24] M. Pajic, I. Lee, R. Mangharam, and O. Sokolsky. Upp2sf: Translating uppaal models to simulink. *University of Pennsylvania, Tech. Rep*, 2011.

[25] M. Pajic, R. Mangharam, O. Sokolsky, D. Arney, J. Goldman, and I. Lee. Model-driven safety analysis of closed-loop medical systems. *Industrial Informatics, IEEE Transactions on*, 10(1):3–16, 2014.

[26] V. L. Patel, V. G. Allen, J. F. Arocha, and E. H. Shortliffe. Representing clinical guidelines in glif. *Journal of the American Medical Informatics Association*, 5(5):467–483, 1998.

[27] R. Pellizzoni, P. Meredith, M. Caccamo, and G. Rosu. Hardware runtime monitoring for dependable cots-based real-time embedded systems. In *Real-Time Systems Symposium, 2008*, pages 481–491. IEEE, 2008.

[28] S. Quaglini, P. Ciccarese, G. Micieli, and A. Cavallini. Non-compliance with guidelines: motivations and consequences in a case study. *Studies in health technology and informatics*, 101:75–87, 2003.

[29] N. Strzyzewski. Common errors made in resuscitation of respiratory and cardiac arrest. *Plastic Surgical Nursing*, 26(1):10–14, 2006.

[30] O. Young, Y. Shahar, Y. Liel, E. Lunenfeld, G. Bar, E. Shalom, S. B. Martins, L. T. Vaszar, T. Marom, and M. K. Goldstein. Runtime application of hybrid-asbru clinical guidelines. *Journal of biomedical informatics*, 40(5):507–526, 2007.