WCIT 2010

An event based formal specification method to diabetic’s behavior monitor system

S.M. Babamir\textsuperscript{a,*} and M. Borhani Dehkordi\textsuperscript{b}

\textsuperscript{a}University of Kashan, Kashan, Iran
\textsuperscript{b}Azad University of Arak, Arak, Iran

Abstract

Modern medical monitor systems shoulder the responsibility of patients’ surveillance. Such life-critical systems steering by software will put patient’s life in peril if they try in vain to satisfy patient’s requirements. As a prelude to reliable software construction of such systems, formal specification of the software behavior has been noted as an effective method because it is susceptible of formal verification. Among other life-critical systems, Continuous Infusion Insulin Pump (CIIP) is a popular medical monitor system that shoulders the responsibility of diabetic’s sugar regulation. This paper aims to present a specification-based formal method to CIIP behavior as a prelude to the CIIP formal verification.

Keywords: Insulin Pump; Patient Monitoring; Formal Specification; Formal Verification

1. Introduction

“Type 1 Diabetes” called youthful sugar and emerges in all ages, even though children, the youth, and ages of before thirty usually are afflicted with the disease. The CIIP system is intended to be worn continuously by a diabetic and the system administers regular doses of insulin based on regular sampling of the wearer’s blood-sugar level. Happened by taking a low-dose, an overdose, or unnecessary dose, monitoring the diabetic in vain is ended to diabetic’s affliction such as cerebral, eye, heart, or kidney diseases. Steered by software, among others medical systems, the CIIP system is one that its behavior should be accurately specified. Such specification leads to constructing reliable software to the system. To this end, we first consider the model we suggested in a recent book [1] and accordingly we specify a formal specification to the system behavior.

Visually illustrated by two Petri-Net automata, the suggested model shows diabetic behavior and CIIP reaction to the diabetic. The first automaton includes diabetic’s high, normal, and low states and transition between them and the second one includes CIIP idle, sampling, computing, and delivering states and transitions between them. CIIP visits diabetic one per ten minutes and makes a transition to its desired state based on the visited state of the diabetic.

* Corresponding Author. Tel.: +98-361-59192485; fax: +98-361-5559930.
E-mail address: babamir@kashanu.ac.ir
When the CIIP system is controlling the diabetic’s blood sugar, the physician expects that the following safety requirements should be satisfied.

(SR1) “Blood sugar never falls below safe-min”, stated as (a),
(SR2) “The diabetic should not overdosed on insulin”, elicited from (b, c),
(SR3) “The system should deliver no unnecessary insulin”, elicited from (c),
(SR4) “The system should deliver no short of insulin”, elicited from (b, c).

There are two other requirements not being the main requirements posed by the physician but imposed by physical constraints of the system.

(SR5) “The system sensor should not be early in sampling”,
(SR6) “The system sensor should not be late in sampling”,
(SR7) “The system reservoir should not consist of insufficient insulin at the time of dose delivery”.

Considering diabetic’s safety requirements, this paper aims to extract formal specification of the CIIP behavior from the automata states and transitions. The specification consists of rules stated by Relation 1. The relation states that CIIP should be held at state $\sigma$ if event $\alpha$ happens where $\alpha$ indicates that diabetic makes a transition from its current state into some new state.

$$\alpha(\sigma, \tau) \rightarrow \text{happen}(\alpha, \tau) \land \text{trigger}(\alpha, \sigma) \tag{1}$$

2. Specification of system behavior of an insulin pump with a timed Petri Net

One of the problems to which diabetic patients are faced is that when their blood glucose rises they need an injection of a specified dose of insulin. The CIIP system is the system aimed to automate automated injection where the system every ten minutes takes a sample of the patient’s blood and performs some computations on it. In the event of the patient needing insulin injection, the system injects the appropriate dose to patient’s body. It should be always kept in mind that errors arising in such a system might place the patient’s life at risk. We provide a model for the system behaviour via Petri Net.

We define the Petri-Net of the CIIP system (depicted by Fig. 1) by the 5-tuple $(P, T, F, W, M_0)$ where, $P=\{P_{\text{Diabetic}}, P_{\text{System}}, P_{\text{Reservoir}}\}$ where $P_{\text{Diabetic}}=\{P_1, P_2, P_3\}$, $P_{\text{System}}=\{A_1, A_2, A_3, A_4\}$, and $P_{\text{Reservoir}}=\{P'_1, P'_2\}$, $T=\{T_{\text{Diabetic}}, T_{\text{System}}, T_{\text{Reservoir}}\}$ where $T_{\text{Diabetic}}=\{T_1, T_2, T_3, T_4, T_5\}$, $T_{\text{System}}=\{T'_1, T'_2, T'_3, T'_4, T'_5\}$, and $T_{\text{Reservoir}}=\{T''_1, T''_2, T''_3\}$, $W:F \rightarrow 1$, and $M_0=\{M_0(\text{Diabetic}), M_0(\text{System}), M_0(\text{Reservoir})\}$ where $M_0(\text{Diabetic})=\{1,0,0\}$, $M_0(\text{System})=\{1,0,0,0\}$, and $M_0(\text{Reservoir})=\{1,0\}$, i.e. $P_1=1, A_1=1, \text{and } P'_1=1$.

---

Fig. 1. The CIIP System Behavioral Model [1].
Initially, the diabetic is in the "High" state, which may: (1) stay in the same state or (2) change to the "Normal" state. According to physician (Section 1), each diabetic's state is a representation of a range of values; therefore, a change in the diabetic's sugar value may lead to the change or no change of the diabetic's current state. Accordingly, each transition is considered as a mapping from its input arc denoting an input value (shown by "a") into its output arc denoting an output value (shown by "b"). The mappings associated with Petri-Nets transitions of the diabetic and the system reservoir has been expressed in Fig 1. Transitions $T_1$ and $T_3$ in Fig. 1, for instance, indicate that although the sugar value increases or decreases, the value is still high/normal. However, Transition $T_2$ in Fig. 1 indicates that a decrease in the sugar value leads to the change of the state. The "Low" state being a critical state cannot be managed by the system and so it may not be changed to the "Normal" state by the system. This state will appear if the diabetic is overdosed or unnecessarily dosed up by the system.

The system is initially is in the "Idle" state and will change to the "Sampling" state (i.e. $T_1$ will fire) when the system timer generates an interrupt event (indicated by Int.). The system will return to the "Idle" state from the "Sampling" state (i.e. $T_2$ will fire) when it identifies no increase in the diabetic's blood sugar and will change to the "Computing" state (i.e. $T_3$ will fire) when it identifies an increase. The system will change from the "Computing" state to the "Delivering" state (i.e. $T_4$ will fire) when the system reservoir has sufficient insulin. The system reservoir will stay in the "Sufficient" state while the computed dose is available to deliver [1].

3. Specification of safe behaviour of the CIIP system

In the previous section, only the system operations were described paying no attention to its being safe or unsafe. Therefore, to better lucidity and intelligibility of the system's behaviour, the Petri Net visual method was used. However, it is necessary to distinguish safe cases of unsafe ones, so that appropriate decisions could be taken to deal with such situations. Therefore, in this section we will deal with this crucial state employing the state graph.

In safety issues regarding diabetics there are bad events which should not occur. In other words one should not deviate from safety requirements. Therefore, we should specify the CIIP system behaviour leading to violations from safety issues. In this section, we draw the reachability graph (Fig. 2) for the Petri net. Since each of the tokens of the above-mentioned system can move from one location to another at various time conditions, a set of numerous and various states will be provided for the system. Now, we deal with unsafe paths leading to entry of the above said safety issues. Then using the graph states, requirements rules stated in Section 1 will be extracted according to Relation 1 so that the system does not enter unsafe paths.

States $P_1$, $P_2$ and $P_3$ the system states and $P'_1$ and $P'_2$ are the Reservoir ones and X indicates an unspecified state. We show that each state combination is in the form of $s_1$, $s_2$, $s_3$ such that $s_1$, $s_2$ and $s_3$ respectively belong to the Diabetic, system and the system reservoir. In Fig. 2, first the patient's blood glucose is assumed to be high ($p_1$), system in the inactive state (A1) and reservoir is assumed to contain sufficient insulin ($P'_1$). Therefore the reachability graph is drawn from this point. In Table 1(a), based on safety requirements raised in introduction section, and hazardous behaviour identified in the reachability graph, rules will be extracted according to Relation 1 such that these rules warrant integrity of the CIIP system behaviour at run time. To extract the rules presented in Table 1(a) consistent with Relation 1, we need events existing in CIIP system for specification of rules; accordingly, we classify them in Table 1(b). To specify rules of Table 1(a), we have used "~" as negation. Rules mentioned in Table are appropriate to hazardous states in the reachability graph such that if these rules are continuously reconsidered during the run time, it will prevent the system to enter into hazardous states.

4. Implementation of the Insulin Pump system

In this section, employing the C# programming language we will proceed to implement the system behaviour from the Petri Net model and the system monitor based on the specifications mentioned in Table 1(a). Note that due to programming complexities we will restrain to deal with details and will suffice to mention an example of the code. As it can be seen in code 1(b), the system's behaviour has been implemented from the Petri Net model where the system's monitor called Monitor() has been called. Through calling of the get sample() method, the system enters the sampling state. Following sampling, through calling the compute() method, the system enter the computation state.
If the Monitor() returns the value zero, it means that a computation error has happened and the system's operations should be stopped. The while loop verifies that if sufficient insulin exists in the reservoir, insulin delivery should be done; otherwise the system should remain inactive from this state. If the system identified the need to inject insulin, through calling the inject() method, it enters the injection state. Finally through calling the update() method, the system enters the update state and through calling the wait for Next period() method it waits until the next time interval for sampling arises.
private int MONITOR ()
{
    if (sample < safeMin)
    {
        if (injectionDose != 0)
        {
            systemStatus = "error";
            output = "There are some computing errors.";
            return (0);
        }
    }
    else if (sample >= safeMin && sample <= safeMax)
    {
        if (injectionDose != 0 && injectionDose != 1)
        {
            systemStatus = "error";
            output = "There are some computing errors.";
            return (0);
        }
    }
    else if (sample > safeMax)
    {
        if (injectionDose > maxSingleDose)
        {
            systemStatus = "error";
            output = "There are some computing errors.";
            return (0);
        }
    }
    else
    {
        return (1);
    }
}

private void operation()
{
    systemStatus = "runningAutomatic";
    getSample();
    compute();
    if (MONITOR() == 0)
    {
        return;
    }
    while (insulinAvailable < injectionDose)
    {
        outputText1 = "Amount of insulin in resource is very low."
        systemStatus = "error";
    }
    systemStatus = "runningAutomatic";
    if (injectionDose != 0)
    {
        Inject();
    }
    update();
    waitForNextPeriod();
}

Code 1(a) is the system’s implementation monitor and has been written based on specification presented in table 1(a), i.e. it takes care that no unauthorized dose is injected to the patient and verifies the permitted dose considering the blood sugar level. Specification of global variables defined in codes 1(a) and (b) are presented in tables 2. Table 3 shows result of implementing the CIIP system where the maximum injected dose is four units in each dose. Also, the amount of injected units in each dose depends on the two previous sampling results. In this table, results are presented in columns Info, Warning, and Error such that the column relates to the injected dose to the patient based on his/her blood glucose level. The Warning column relates to the time when the patient’s blood glucose is less than normal values and the column Error relates to errors happening to the system such as: Error occurring in the insulin reservoir, error arising in the sampling sensor, error arising in the iSATIN system, error occurring in the device batteries, and etc.

5. Conclusions and Related Work

Since the CIIP system undertakes controlling the diabetic’s blood sugar, the physician expects that the some safety requirements should be satisfied. To this end, based on the model we suggested in a recent book [1], this paper presented a specification-based formal method to CIIP behavior that can be used to the CIIP formal verification. Visually illustrated by two Petri-Net automata, the suggested model showed diabetic behavior and CIIP reaction to the diabetic. The first and the second automata include Diabetic and CIIP states and transition between them respectively. In constrast with some other methods, this paper made contribution, specification of high-level safety requirements by physician language and specification system by a visual modeling. The first case, i.e. the high
level specifications are intelligible to the physician. In [9], Wang et al have just addressed modeling the Insulin Pump behavior using the SOFL formal language. Similarly, Sommerville has used the Z formal language to specify the system and the safety requirements [10]. The second case, i.e. using a visual method made the benefit of the lucid understanding of the system behaviour than the textual methods such as those applied in [4, 5].

References

1. S.M. Babamir. A Framework for Specifying Safe Behavior of the CIIP Medical System. Book Chapter in Advances in Experimental Medicine and Biology, Vol. 696: Software Tools and Algorithm for Biological Systems, Springer-Berlin, ISBN: 978-1-4419-7045-9, Release date: November, 2010.
2. A.Bobbio, M.Gribaudo and A.Horvath, Modeling a Car Safety Controller in Road Tunnels using Hybrid Petri Nets, In Proceedings of 9th International IEEE Conference on Intelligent Transportation Systems, ICTS2006, 2006.
3. K.Fukuzawa and M. Saeki, Evaluating Software Architectures by Colored Petri Nets, In Proceedings of the 14th international conference on Software engineering and knowledge engineering, ACM International Conference Proceeding Series: 27: 263-270, 2002.
4. S.Mo Cho, H.Seok Hong, and S.Deok Cha , Safety Analysis Using Colored Petri Nets, In Proceedings of Asia-Pacific Software Engineering Conference (APSEC-96), pp: 176-183, 1996.
5. K.Buchacker , Combining Fault Trees and Petri Nets to Model Safety-Critical Systems, In Proceedings of High Performance Computing Symposium, pp: 439-444, 1999.
6. K.Sacha , Safety verification of software using structured Petri nets, In Proceedings of the 17th International Conference on Computer Safety, Reliability and Security, pp: 329-342, Springer, 1998.
7. R.David and H.Alla, Discrete, Continuous, and Hybrid Petri Nets, Springer, 2005.
8. C. Ghezzi. et al Fundamentals of software engineering, the second edition. Prentice-Hall, 2003.
9. J. Wang et al. Developing an Insulin Pump System Using the SOFL Method. The 14th Asia-Pacific Software Engineering Conference, pp. 334-341 2007.
10. I. Sommerville. Software engineering, the 8th Edition, Addison Wesley (2006)