Poster Abstract: Power Usage of Time and Event-Triggered Paradigms: A Case Study

Taylor Johnson, Sayan Mitra
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801 USA
{johnso99, mitras}@illinois.edu

***Abstract*—We evaluate the time-triggered (TT) and event-triggered (ET) programming paradigms in the context of developing large-scale distributed real-time systems with fault-tolerance. To this end, we present a simple case study using the Lego Mindstorms NXT platform in an intrusion detection problem, and compare the power consumption and accuracy of event detection in TT and ET systems. We use this case study comparison as part of the motivation for the development of a new mixed ET and TT language.**

I. Introduction

For hard real-time systems, timing guarantees mandate a critical part of control system development. Time-triggered (TT) and event-triggered (ET) architectures have been studied with regards to scheduling and reactivity for some time [1]. The TT programming paradigm Giotto [2], and its extension to hierarchical timing language (HTL) [3], are the first software solutions that do not also specify an underlying architecture, accomplished by utilizing the concept of logical execution time (LET) to make timing guarantees. We are interested in similar software solutions. With LET, the programmer specifies at what time outputs of a task are available, and a compile-time check of this specification determines if it is implementable on a given platform [2]. The event-driven xGiotto language [4] relieves the limitations on asynchronous event handling that TT languages suffer from, but its schedulability becomes intractable.

For our case study, we consider a potentially rare and short event occurrence. The only way for a TT system to detect the event is to utilize a short LET period for a task. This high-frequency task restricts the programmer from effectively utilizing a processor's sleep capabilities, resulting in increased power consumption, as data movement must occur every LET period. Furthermore, to detect this event, the underlying real-time operating system (RTOS) may need to perform a context switch each time this event is detected, leading to a lower bound on the period in which this task can execute that is higher in TT than ET.

The main observation of this work is that TT paradigms at the software-solution level of abstraction consume more power than ET paradigms. We presented a case study intrusion detection problem and observed a roughly 20% average power savings in an ET paradigm versus TT as shown in Table I.

II. Case Study

Figure 1 shows the intrusion detection setup. The guard detects if the intruder is entering the guarded region when the beam of reflected laser light is broken. Upon detecting the intruder, the guard signals an actuator, the alarm. The primary constraint on whether the intruder is detected is the relationship between the rate at which the sensor is queried and the speed of the intruder, such that, if the intruder moves faster than the time between consecutive LET periods, the intruder will be undetected. Missing of events will occur in the TT paradigm when the LET period is above the rate at which the events occur. (The Nyquist-Shannon sampling theorem is satisfied by the lower-level hardware sampling, rather than the virtual sampling here.) There are two types of events to be detected, on and off events, and there are two possible durations for these events, $300ms$ or $5000ms$. The events are generated by the intruder, which uses an actuator to block or unblock the light source, over a sequence of 100 events that was randomly generated and then fixed before the comparison.

The Lego Mindstorms NXT platform utilizes an ARM7 microprocessor, a microcontroller that handles analog-to-digital conversion (ADC) and

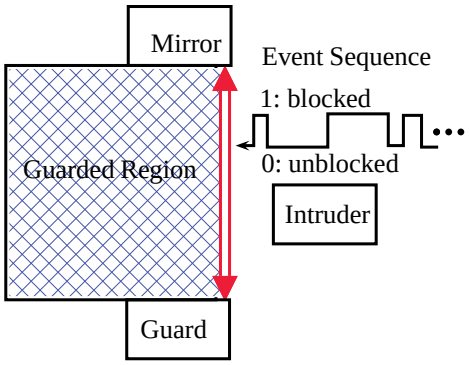

Fig. 1. Intrusion Detection Problem Setup

digital-to-analog conversion (DAC) of modular sensors and actuators, and communications capabilities through Bluetooth. The open-source nxtOSEK RTOS is used. The system is powered from a DC power supply, with a shunt resistor inserted on the low-side output of the supply for current measurement.

We have implemented two versions of an embedded system that implements this detection. One version is TT, effectively polling with a task the digitized sensor values to see if they have changed every T seconds, the LET period, while the other version is ET, and waits in idle-mode for a change in the digitized sensor values, detecting the event after at most one hardware sampling period ($3ms$). The TT version utilizes the E-machine of [5] ported to nxtOSEK, whereas the ET version relies on the OSEK notion of events utilizing a concurrent bounded FIFO queue whose correctness is proved in [6].

For several runs of the same randomly generated trace of events described above, there is a 20% average power savings in the ET paradigm versus TT as in Table I. The error percentage represents the amount of misclassified events, that is, short detected as long and vice-versa. Missed events are the number of events completely undetected.

III. Future Work

Our long-term goal is to create a new programming language and supporting runtime environment that handles TT and ET tasks, aimed especially at fault-tolerant distributed applications, such as sensor networks or multi-agent systems. To represent the formal semantics of our new language and its interaction with its context and environment (RTOS, etc.), we plan to model it using Hybrid Input/Output Automata (HIOA) [7]. The Embedded Machine [5] provides platform independence, and we plan to use a similar concept for our runtime environment.

TABLE I
Event and time-triggered paradigm comparison of power consumption and event detectability

Paradigm	ET	TT (150ms)	TT (300ms)	TT (450ms)	TT (1000ms)
Average Power (W)	0.87	1.10	1.10	1.11	1.08
Missed Events (#)	0	0	0	5	44
Error (%)	0.08	0.08	0.08	0.20	1.10

IV. Conclusion

Our observation is that TT systems consume more power than equivalent ET systems, at the level of abstraction software solutions mandate. This perhaps is an obvious result when considering polling versus interrupts, but the empirical evidence had not previously been established for this higher level of abstraction. The decreased reactivity to events in TT languages due to sensor inputs being frozen at the beginning of the fixed LET period leads to increased power consumption when attempting to detect rare events, as the processor must perform data movement operations at a high rate. For this and other reasons, we would like work towards a TT and ET language so that we have increased flexibility to deal with such problems as rare and narrow event detection.

References

[1] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Norwell, MA, USA: Kluwer Academic Publishers, 1997.

[2] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," in *Proceedings of the IEEE*. Springer-Verlag, 2001, pp. 166–184.

[3] A. Ghosal, T. A. Henzinger, D. Iercan, C. Kirsch, and A. L. Sangiovanni-Vincentelli, "Hierarchical timing language," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-79, May 2006.

[4] A. Ghosal, T. A. Henzinger, C. M. Kirsch, and M. A. A. Sanvido, "Event-driven programming with logical execution times," in *Proc. of HSCC 2004, Lecture Notes in Computer Science*, 2004, pp. 357–371.

[5] T. A. Henzinger and C. M. Kirsch, "The embedded machine: Predictable, portable real-time code," *ACM Trans. Program. Lang. Syst.*, vol. 29, no. 6, p. 33, 2007.

[6] C. Gong and J. M. Wing, "A library of concurrent objects and their proofs of correctness," Carnegie Mellon University, Tech. Rep., 1990.

[7] N. Lynch, R. Segala, and F. Vaandrager, "Hybrid i/o automata," *Inf. Comput.*, vol. 185, no. 1, pp. 105–157, 2003.