Verification in Networked Embedded Systems [*]

Fang Yu
Department of Computer Science
University of California, Los Angeles

August 31, 2006

1 Abstract

In this summer, we propose to automatically translate nesC to hybrid/timed automata, which could be served as the first step to achieve Networked Embedded System verification. Based on that, we may verify some ongoing projects such as classification. Some verification issues, e.g., scalability, are also discussed.

2 Motivation

Compared to test or simulation, formal verification poses an attractive way by either guaranteeing the correctness of the system or generating a counter example to indicate the violation. However, due to state explosion, the power of formal verification is usually restricted to trivial systems.

This project is motivated by two observations of Networked Embedded Systems: a) due to limited physical resources, device's behavior is simple and is implemented in elegant codes, and b) usually all devices are the same, i.e., the system is highly symmetric. These yield two advantages while adopting formal verification: a) a concise formal model may be achieved and b)symmetry reduction may be applied. Both alleviate state explosion problem. In this manner, large-scale system may be verifiable.

3 Networked Embedded System verification

There are two verification levels while considering networked embedded systems: unit verification and system verification. For unit verification, we make sure that each individual device works properly with the well-defined interface interacting the environment,

[*]This is the draft for the summer research with Prof. Sarrafzadeh, Summer06

while in the system verification level, we focus on the correctness of the integrated system under the correct individual behavior.

I think three issues arise here: a) Investigate standard/efficient mapping rules to hybrid/timed Automata for defined nesC statements, and b) Optimize the generated automata and c) Scale the system size.

4 Unit verification

To achieve unit verification, we need a) a formal model for each device and b) specifications for desired properties. Both should be generated formally and automatically. We propose to construct a well-defined translator for some standard language of embedded systems. We aim to automatically translate *nesC* [8] to Linear Hybrid Automata(LHA) [3] or Timed Automata(TA) [1]. nesC is a specific programming language for networked embedded systems, which incorporates component-oriented design, event-driven execution, and concurrent model. Basically, nesC applications are built out of components with well-defined, bidirectional interfaces.

A LHA is a finite-state automaton equipped with a finite set of dense system variables which can hold real-values, while a TA could be viewed as a LHA but only equipped with real-time clocks, which had been widely adopted to verify real-time systems in modern model checkers [13, 17, 21].

Some issues here are a) how to determine/define the exact semantics of nesC and its corresponding modules of timed automata, b) how to convert even-driven semantics to real-time semantics, and c) how to represent interfaces/environments. The parts of these issues had been addressed in [19], in which we translated timeC, a C-like language with time statements, and OVL assertions to the input language of RED [17].

5 System verification

In this verification level, we focus on the integrated behavior correctness. Precisely, we want to determine whether the integrated system behaves well under the assumption that each device behaves well(as you expect).

To achieve system verification, we have to answer both questions first: a) how to connect each device together, e.g., using communication channels(e.g., lossy channels) and b)how to implement specific algorithms for your purpose in a distributed manner, e.g., classification. Basically, all these should be implemented using nesC. In other words, once we determine and implement the environment and algorithms in nesC, we should be able to automatically generate the corresponding timed automata.

After constructing the timed automata, the next step is to discover the desired properties of the system. These specifications should be described in formal logics, e.g., temporal computation tree logics(TCTLs).

One of the most important issue in networked embedded systems is *reliability.* Basically we call in *Fault tolerance verification.* As Foad mentioned, assume we have fault tolerant technique in a distributed embedded system that claims to tolerate loosing 1 (or more) PE(Processing Elements, such as Mica2Dots). How can we specify the property and verify the functionality? Can we describe it either as an explicit TCTL formula or as a set of assertions inline in nesC codes? It remains unclear to achieve Fault tolerance verification so far, but it appears to be an interesting issue.

With Timed automata and specifications, we can verify the system, i.e, whether the target system satisfies the desired properties. To achieve this, a couple of model checkers with different data structure can be used e.g., RED(BDD-like) [17], xBMC(SAT-based) [21], and UPPAAL(DBM) [13]. RED is a full symbolic TCTL model checker with plausible performance. Hence, I tend to choose RED as our checker.

6 Scalability

To verify large systems, we first have to conquer the state explosion problem, i.e., the searching space is exponential blow-up while increasing the size of the system. Being capable of verifying small systems well, the remain challenge is how to scale our verification result to large systems. Can we infer the correctness of a large system by verifying a set of specific small/verifiable systems? Various reduction techniques, such as partial order reduction and symmetry reduction, had been proposed to address this problem for general cases. It appears to be attractive but remains unclear how these techniques affect verification performance for networked embedded systems, a large but highly symmetric environment.

Another approach is induction. Can we construct an induction proof such that after proving the base (by verifying basic systems) and the induction step(induction on the system size), the correctness of arbitrary large systems hold? It remains unclear how to define the induction step and construct the induction proof. Some properties of verification algorithms, such as termination, soundness and completeness may alos worth to be further addressed. We call in *Inductive verification.* Induction technique had been used to accelerate fixed point calculation in [20]. In [14], Namjoshi showed, for general temporal properties, how to lift a deduction proof of an abstract program to the original one. Similar techniques may be applied here.

Considering reduce the complexity, another idea is modularity. Recall that our target software is embedded, distributed onto networks and structured into logical components that interact asynchronously. Can we abstract the system, e.g., treating one thousand nodes as one with a new interface, such that both systems have the same behavior with respect to the desired properties? We do not necessary *abstract* systems. The key point here is looking systems in different levels, such that in each level we focus on specific properties and hence are able to ignore the details in lower levels. An idea is modelings devices in different levels such that we can do unit verification and then achieve system verification. Some variation here is adjusting each unit size and investigating the relationship between units and the system(could be treated as a large unit). Thus, we might construct an induction proof based on the unit size. We call in *Modularity verification.* Alur et al. first proposed the idea of modularity in [4]. They further realized their ideas in the model checker MOCHA for discrete system verification. Broy [5] introduce a comprehensive mathematical model for composed systems, its essential views and their relationships Models of data, states, interfaces, functionality, hierarchical composed systems, and processes are considered. Some issues in abstraction and refinement, as well as forms of composition and modularity, had been addressed.

7 TinyOS

- ADCC.nc
- Logger.nc
- ServiceSchedulerM.nc
- ADCM.nc
- LoggerM.nc
- SimpleTime.nc
- AMPromiscuous.nc
- LogicalTime.nc
- SimpleTimeM.nc
- AMStandard.nc
- Main.nc
- TimeUtilC.nc
- BufferedLog.nc
- NoCRCPacket.nc

- TimerC.nc
- ByteEEPROMAllocate.nc
- NoLeds.nc
- TimerM.nc
- CRCPacket.nc
- PacketSink.nc
- UART.nc
- Checkpoint.nc
- PotC.nc
- UARTComm.nc
- ClockC.nc
- PotM.nc
- UARTFramedPacket.nc
- CrcFilter.nc
- RadioCRCPacket.nc
- UARTM.nc
- FramerAckM.nc
- RadioNoCRCPacket.nc
- UARTNoCRCPacket.nc
- FramerM.nc
- RandomLFSR.nc
- Voltage.nc
- GenericComm.nc
- RealMain.nc
- VoltageM.nc

- GenericCommPromiscuous.nc
- Reset.h
- WatchDogC.nc
- I2CPacketC.nc
- ResetC.nc
- sched.c
- I2CPacketM.nc
- SecDedRadioByteSignal.nc
- LedsC.nc
- ServiceSchedulerC.nc

8 Application

In the last section, we target the possible applications for networked embedded systems.

8.1 Power Analysis

One essential requirement for sensor networks is the reliability of applications since sensors are planned to deploy into the area once and unattended for a period of time without maintenance. An interesting focus is the lifetime of the sensor network. In some sense, the lifetime for a hardware configuration reflects the period from the beginning till it out of charge. Since concurrent interactions between tinyOS components may make the behavior of applications hard to predict, it appears to be attractive to adopt formal verification to support power analysis.

In a sensor network, the data being sensed must be transmitted to base station so that the end-user can access the data. To conserve power, the data is usually being relayed multiple times towards destination. Coleri and Ergen [6] performed the power analysis of one node as a function of the distance from the base station. They first estimated energy consumptions of the instruction set in tinyOS, and then modelled each component as a Linear Hybrid Automaton (LHA), such that each component records its power consumption during the computation. The distance is served as a parameter of LHA to adjust the marginal rate of cost. The assumption behind Coleri's model is that the nodes closer to the base station will relay more packets compared to the far away one. While they assume the flow of a node has an opposite relation to the distance between the node and base station, in real sensor networks, the flow usually depends on its routing algorithm. In particular, researchers have proposed many routing algorithms to minimize the power consumption of either the entire system or each node, in a sense to extend the lifetime as long as possible.

Considering these energy-aware algorithms, we extend Coleri's work to perform the power analysis of one node as a function of its flow. Roozbeh et al. [11] proposed an ϵ-optimal multi-hop routing technique such that given the network topology with a set of sources and destinations, the optimal flow, with respect to load balance among nodes, can be calculated in a centralized manner. They reduce the problem to the min-cost problem, and hence achieve polynomial time complexity by solving linear programming.

Incorporating Roozbeh's work, we can serve the optimal flow as the input configuration for each node, and verify whether a given hardware configuration is sufficient to match the workload. In other words, we aim to check whether all nodes have sufficient resources to support the calculated optimal flow. If the answer is no, a counter example is generated to depict how a node fails even for this optimal balanced flow, i.e., the possible best load distribution/the best lifetime of a node. On the other hand, since all nodes are identical, the bottleneck of the flow can be identified easily. Instead of checking all nodes, we may assert that once the node with the heaviest load can take the work, the system will not crash.

We summarize our ideas as below.

- Given a directed graph $G = (V, E)$,$S = (s, q, t)|s \in V, q \in N$, we first calculate the ϵ-optimal flow f.
- Let $f_M = \max\{f(i)|i \in V\}$ denote the max flow into a node. Assume each node is implemented by a nesC program P. We then generate the hybrid system such that the corresponding LHA A could be verified against the risk property, "the node is out of charge before finishing the workload."

In the following week, I will focus on how to generate the hybrid system given the nesC program and the required flow.

8.2 Security Protocol Analysis

Authenticated broadcast for severely resource-constrained environments. [15]

8.3 Classification

Foad suggests one of his ongoing projects: classification. Assume given the classification algorithm on real time data which is implemented in a distributed fashion on PE. Can we verify its correctness, as well as its reliability?

There are three different components needed to be implemented in nesC: a) Software in PE, b) Networking protocols and communication c) Overall Algorithm. Regarding communication, we can pick a well known communication protocol, e.g., [9] for the problem. Classification algorithms can potentially be: tracking objects, image classification, environmental data processing (temperature, humidity...) physiological data classification (ECGK).

Furthermore, it is also interesting to categorize these algorithms and have a model for each category. I interpret this idea as finding some common characters for algorithms in the same class and implement them in the same way. If this is the case, we might say a model can represent a set of algorithms, and the correctness of this model can infer the correctness of all algorithms in this set.

9 Discussion

Alur et al [3] present a model checking procedure and its implementation for the automatic verification of embedded systems. Systems are represented by Hybrid Automata -machines with finite control and real-valued variables modelling continuous environment parameters such as time, pressure, and temperature. System properties are specified in a real-time temporal logic and verified by symbolic computation. This is exactly the procedure we are going to adopt in this project. While their works address embedded systems, we extend their techniques to networked embedded systems.

The first challenge is to formalize the semantics of standard languages which are used to implement networked embedded systems.

Volgyesi et al. [16] presented a modelling environment targeting tinyOS. They propose *Hierarchical Interface Automata*(HIA) as the formalism to capture the temporal and type aspects of interfaces and support the composition and verification of components. They further use the explicit model checker:SPIN to check composability of components based on their interface models, and also verify whether the implementation of a component matches its formal model. However, as they mentioned, their approach suffers from scalability and might be limited to verify entire sensor networks. To address this problem, considering the nature of networked embedded systems, we propose to determine automorphisms and apply symmetry reduction to reduce the complexity of model checking. The basic idea is to reduce model checking over the original structure M to a smaller quotient structure M', where symmetric states are identified. More precisely, the symmetry of M is reflected in the group of permutations of process indices defining graph automorphisms of M. [7]. Symmetry takes advantage of the permutations on the components of a state which provoke the same executions for a specific property. Since we particularly focus on those systems composed of many isomorphic processes, we may be able to gain significant, even exponential, savings in the complexity of model checking.

Despite the scalability, it is also unclear how to analyze continuous behavior in HIA. One interesting research direction is extending HIA to hybrid systems, e.g., constructing Hierarchical Hybrid Automata and developing model checkers to verify them. To my best knowledge, most modern hybrid model checkers, e.g., Trex [2], HyTech [10], and RED [18], seldom address this feature.

Opposite to general purpose, some researchers focus on verifying specific applications. Coleri and Ergen [6] do power analysis of sensor network by modelling tinyOS as hybrid automata with HyTech [10]. With respect to power analysis, an overview of the tinyOS hybrid automata model is proposed, as well as the semantics of the components and the network. Basically, each component in tinyOS is abstract to an automaton containing three states: actual, energy, and wait, such that during the computation, the power consumption of each component is accumulated. They generate a trace with Hybrid for each node to estimate its power consumption. They further simulate muti-hop forwarding in a hierarchy-tree structure network and conclude the relation of distance and lifetime of the sensor nodes. Compared to automatically translate nesC to hybrid automata syntactically, they target the semantics of some specific application and model the system on that purpose.

10 Summary

Finally, for these two months, we aim to target an application which could be implemented in nesC and build the translator which could automatically translate the previous system into LHAs, such that some well-known model checker, e.g., RED, can be used to verify the target system.

References

[1] R. Alur, "Timed Automata." In Proc. of CAV'99, LNCS 1633, pp. 8-22, Springer, 1999.

[2] A. Annichini, A. Bouajjani, and M. Sighireanu, TReX: A Tool for Reachability Analysis of Complex Systems, In Proc. of the 13th Intl Conf. Computer Aided Verification, pp. 368-372, 2001.

[3] R. Alur, T. A. Henzinger, P.-H. Ho, Automatic Symbolic Verification of Embedded Systems. In Proc. of Real Time Systems Symposium. IEEE Computer Society Press, 1993.

[4] R. Alur, T. A. Henzinger, F. Mang, S. Qadeer, S. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. In Proc. of the Tenth Int. Conference on Computer Aided Verification, LNCS 1427, pages 521-525. Springer-Verlag, 1998.

[5] M. Broy, Modular hierarchies of models for embedded systems, Formal methods and models for system design: a system level perspective, pages 3-32, 2004.

[6] S. Coleri, M. Ergen, Verification and Power Analysis of an Event-Based System (TinyOS) and Sensor Network with Hybrid Automata. SCI Orlando, July, 2002.

[7] E. A. Emerson and A. P. Sistla, Symmetry and model checking, In Proc. of the International Conference on Computer Aided Verification (CAV93), LNCS 697, pp. 463-478, Elounda, Greece, 1993.

[8] D. Gay, P. Levis, R. Behren, M. Welsh, E. Brewer, and D. Culler, The nesC Language: A Holistic Approach to Networked Embedded Systems. In Proc. of Programming Language Design and Implementation (PLDI) 2003, June 2003.

[9] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, Energy-Efficient Communication Protocol for Wireless Microsensor Networks, In Proc. of the 33rd Annual Hawaii International Conference on System Sciences(HICSS), pp. 3005-3014, Jan. 2000

[10] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi, HyTech: A Model Checker for Hybrid Systems, Software Tools for Technology Transfer, vol 1, pp. 110-122, 1997.

[11] Roozbeh Jafari, Foad Dabiri, Majid Sarrafzadeh, ϵ-Optimal Minimal-Skew Battery Lifetime Routing in Distributed Embedded Systems, In Proc. of the Journal of Low Power Electronics (JOLPE), vol 1, no. 2, pp 97-107, September 2005.

[12] R. P. Kurshan, V. Levin, M. Minea, D. Peled and H. Yenigun, Static Partial Order Reduction, In Proc. of Tools and Algorithms for Construction and Analysis of Syatems(TACAS), LNCS 1384, pp. 345-357, Lisbon, 1998.

[13] K. G. Larsen, P. Pettersson, and Y. Wang, UPPAAL in a Nutshell, In Int. Journal on Software Tools for Technology Transfer 1(1-2), pp. 134-152, 1998.

[14] Kedar Namjoshi, Lifting Temporal Proofs Through Abstractions, In Proc. of the Fourth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI03), New York, January 9-11, 2003.

[15] A. Perrig, R. Szewczyk, V. Wen, D. Culler, J. D. Tygar, SPINS:Security Protocols for Sensor Networks, In Proc. of Mobile Computing and Networking, Italy, 2001.

[16] P. Volgyesi a, M. Maroti b, S. Dora b, E. Osses b and A. Ledeczi Software Composition and Verification for Sensor Networks. Science of Computer Programming, Vol. 56 , Issue 1-2, pages 191 -210, April 2005.

[17] F. Wang, Efficient Verification of Timed Automata with BDD-like Data-Structures. VMCAI'03, LNCS 2575, Springer-Verlag, 2003.

[18] F. Wang, Symbolic Parametric Safety Analysis of Linear Hybrid Systems with BDD-like Data-Structures. In Proc. of the Sixteenth Int. Conference on Computer Aided Verification (CAV04), LNCS 3114, Springer-Verlag; Boston, USA, July 2004.

[19] F. Wang and F. Yu, OVL Assertion Checking of Embedded Software with Dense-Time Semantics, In Proc. of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA 2003), LNCS 2968, February 2003.

[20] F. Yu and B.-Y. Wang, Towarded Unbounded Model Checking for Region Automata, In Proc. of the 2nd International Symposium on Automated Technology for Verification and Analysis (ATVA 2004), LNCS 3299, pages 20-33, Taipei, Taiwan, Oct 2004.

[21] F. Yu and B.-Y. Wang, SAT-based Model Checking for Region Automata, Submitted to the International Journal of Foundations of Computer Science (IJFCS), 2006.