PRACTICAL METHODS FOR SELF-TESTING OF IoT EMBEDDED SYSTEMS IN ADVANCED MANUFACTURING

Iliya Georgiev, Ivo Georgiev

Department of Mathematics and Computer Sciences
Metropolitan State University of Denver
e-mails: {gueorgil, igeorgi1}@msudenver.edu
USA

Abstract: Advanced manufacturing extends the scope of new challenges in validation of the Internet of Things (*IoT*) based embedded systems that control the manufacturing cells. The paper presents some approaches of self-testing oriented to keep the IoT systems in predictable stable state. The key technology of the discussed self-testing is simulation of the basic functions. The simulator generates some stimuli and compares the responses with some etalon or simulated results.

Key words: IoT-enabled manufacturing, simulation-based self-testing, XML stream, timing testing.

1. INTRODUCTION

Manufacturing on different levels becomes more computerized by continual integration of embedded systems, which are connected in local area networks (LAN) and open to the world by Internet (IoT) [1]. The kernel of such automation is the integration between design, development, interoperability and coordination in cyber-physical systems, further extended to cloud-based manufacturing. The incorporated embedded systems have significant computational power to run a full range of processes, from hard real-time to batch ones. Inter-process communication in the entire environment (manufacturing cell, local network, design centre, cloud) is based on message passing services.

Advanced manufacturing has various industrial and computer components (some of them legacy) that exchange very sensitive data [2]. The manufacturing equipment is influenced by several noises: internal noises (cross talk noises, mutual interference noises, impulse noises, thermal noises) and external influence (electrostatic discharge, electromagnetic stress, mechanical vibrations). On the other side, Internet connection makes the cells target of diversity of adversaries. Some of them could be declared as classical security attacks like viruses, side-channel attacks, node replication attacks and so on. New challenges are the time attacks that can manipulate the timing characteristics on different levels of processing.

The objective of the presented methods is to check dynamically the functional stability of the manufacturing resources. We consider a manufacturing cell that is controlled by IoT embedded systems connected through a time-sensitive local network. The paper presents practical approaches to self-testing based on simulation.

The structure of the paper is as follows. Section 2 defines a prototypical model of the IoT-enabled manufacturing and declares the self-testing scenario. Section 3 explains the suggested self-testing approaches: proving the stable state; self-testing of the IoT embedded systems; timing analysis; XML transaction stream for data exchange; web service simulation by transactions; emergency reboot after some unpredictable unsteady functionality. Section 4 discusses the main contributions and lays out our plan for further development in the area of chaos engineering. Section 5 concludes the paper.

2. PROBLEM DEFINITION

2.1. A Model of IoT-enabled manufacturing

In this section, we lay down a prototypical model of factory automation and discuss the functionality stack (Figure 1). Every cell has sensors and actuators that are connected to different machines, robots, automatic test systems, quality assurance instruments, 3D printers, etc. The cell is driven by an IoT-enabled embedded system that has a variety (several dozens) peripheral devices: parallel ports, serial ports, timers, digital-to-analogue converters, analogue-to-digital converters, synchronized and hand-shaking interfaces, analogue comparators, power width modulators, USB, Ethernet and Controller Area Network, etc.

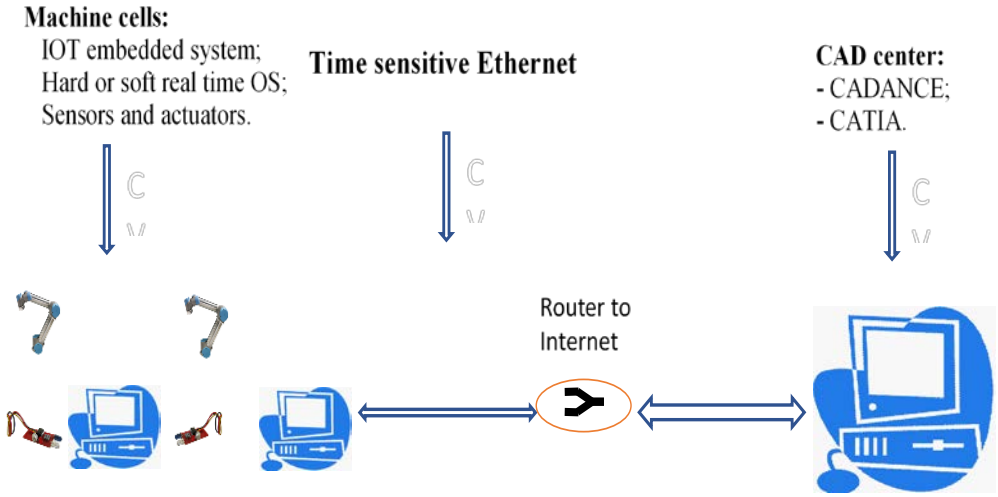

Figure 1. A Model of IOT-enabled manufacturing

The peripheral devices make request for processing by interrupts that are manipulated by internal interrupt controller. Every peripheral device has interrupt priority, mask and a separate vector address for its interrupt handler.

The embedded operating system (OS) can manage hard and soft real-time multithreading processing. Here are some unique characteristics that are different from the conventional OS:

a) I/O operations are not under control of the OS; effectively no device needs to be supported by the OS, except the system timer.

b) Interrupts are not restricted to the OS; every accepted interrupt calls directly the corresponding interrupt handler;

c) Protection mechanism between threads is not supported by the OS; the application threads can execute all instructions including privileged ones.

The embedded systems are connected in a LAN that is running a time-sensitive extension of Ethernet. Every IoT controller is an Internet host and supports application specific protocols for message exchange. The most popular technology are web services.

The information for the manufacturing is generated by the computer aided design (CAD) centre. Typical design automation packages are CATIA for mechanical engineering and CADANCE for electronic design. CAD centre is not connected to the IoT network. It exchanges XML (Extended Mark-up Language) data with the IoT embedded systems via Internet.

In advanced manufacturing, the documentation is presented in XML vocabularies of the engineering standards. Leading is the AutomationML (Automation Mark-up Language) [3], which is an XML vocabulary for engineering information of manufacturing objects (aggregates and details) in their different areas, e.g. mechanical engineering, electrical design, robot control, product lifecycle management. It can describe small details, or a drone or a complete manufacturing cell. Typical standard objects are oriented to the topology, geometry, kinematics and logic.

In all engineering areas the classical standard STEP was converted to STEP-XML [4], which is widely used in aerospace, automotive, shipbuilding and electronic industry. It provides different application protocols: for configuration-controlled mechanical assembly design, for finite element analysis models, for electronic/mechatronic design information and others.

The suggested information-exchanging technology is an XML stream that is circulating through the connected computers [5]. The stream is divided into XML elements with different encryption containing metadata for access control and references to production procedures and data. The stream elements are oriented to some IoT cell and have references to procedures that are concrete industrial hierarchy objects. Every element has one authorized author (writer) and several client (readers).

IoT application process can use web services to get the stream but can decrypt only the sections that are oriented to its production role. After authentication and

proof of permissions, the client software can open the metadata and parse it for availability of the needed version of the specific procedures. In result, the client has open access to the required document database for the duration of the session.

The message-passing technology between the IoT embedded systems and other connected computers is by web services that allow communication between processes that run in different program environments. Web service composition objects in terms of languages and tools are well classified in [6].

The presented model of adaptable computer-aided and configurable manufacturing is service-oriented and is securely open to the cloud.

2.2. Self-testing Scenario

Testing of IoT-enabled manufacturing meets specific challenges related to heterogeneity, real time processing, interoperability etc. (recommended reading is [7]). The application programs that run in the IoT embedded systems are considered correct to control the production and to meet the worst-case execution time in real-time mode, but some generated by the environment unexpected events can put the manufacturing cell in unsafe and unpredictable state.

r that are organized in the following functional groups:

a) Stabilization procedures bringing the peripheral devices, ports, and operating system in some well-known state;
b) Checking of the timing restrictions in the cell and network;
c) XML stream analysis;
d) Web services testing as transactions;
e) Micro-rebooting.

3. PROBLEM SOLUTION

3.1. Testing of the Stable State of the IoT Embedded System

An IoT embedded system runs several threads, some of which are real-time that must meet deadlines. The key to the testing is to prove the stable state of the embedded system (stabilization) that is a recommended step before testing because it controls exactly what kind of operations are under execution. Stabilization requires fixing the inputs in some initial state so that the threads can be executed repeatedly generating predictable outputs. We run a simulator subroutine that stabilizes all the inputs. It can reproduce the exact inputs repeatedly. Once the system is stabilized, when additional tests are activated, there is a guarantee that the changes in the outputs are a function of the new test operation and not due to the input changes.

We define the following steps for stable state testing.

First, the IoT embedded systems are initialized by a well-defined procedure, activated when the production cycle is interrupted by a change in the instrument state or by a scheduled maintenance. The kernel of the operating system has been initialized and is running. The interrupt handlers are accessible. The peripheral devices are in a reset state. No ports are initialized.

Second, the ports and the peripheral devices are activated in running state without any processing. The vector addresses of the interrupt handlers are compared and verified. Send and receive functions are invoked with dummy parameters.

Third, the execution speed of an embedded system is determined by a clock control with the option of programmatic change to the main clock frequency. Reducing the main and bus clock frequency requires less power to run and generates less heat. Speeding up the clock allows higher instruction rate per second and requires more operational power. Part of the stable state analysis is to simulate the clock control by cycling over all possible frequencies.

Lastly, the Internet connection is verified by checking the network address translation, which assigns dynamic IP addresses. The simulator organizes a bouncing mode for IP address translation and checks the possible tunnelling between IPv4 and IPv6 formats [8]. Additionally, it verifies the internal path tables and DNS (Domain Name Service) support. Communication with CAD computers is verified by handshaking web services by exchanging the roles of clients and servers.

3.2. Simulation of the IoT Embedded System and Timing Testing

Timing is important characteristics of the entire computing configuration from the sensors/actuators through the OS to the programming threads to the Internet. The timing behaviour of all items must be predictable, and some upper bounds of the latencies must be documented. The OS must provide precise physical (astronomical) time services like scheduling and memory management.

IoT-enabled manufacturing requires timing between the Internet messaging and the factory automation networks. The dominating trend is to implement time-sensitive networking, which is intended to address the timing and synchronization needs of embedded systems with standard Ethernet technology [9]. All connected computers need to have common synchronized time and to do real-time communication by standardized reserving of bandwidth and time intervals.

The accepted testing approach follows the next scenario. The IoT embedded systems are tested in pairs. The pair consists of a system under test that runs its real-time applications and another embedded system as a ‘tester” of the first one. The selection of the roles is random.

The testing unit initializes communication with guaranteed end-to-end latency. It supports a closed loop control that is operating between both IoT embedded systems. The communication over the Ethernet network is separated into fixed length repeating time slices. Within these cycles, it is possible to prioritize those traffic items that can't be pre-empted. By occupying the network for specific time periods, time-critical communication can be separated from non-critical background transmissions. The time-critical traffic can be transmitted without interruptions.

The testing contains the following procedures.

Check of the common time of the embedded system under test. For real-time communication with hard restrictions to the transmission delays, all network

IoT cells must have a common time reference and synchronize their clocks among each other. This is not only true for the end-to-end devices of the network, such as IoT embedded systems, but also valid for the network component switches. System-wide clock synchronization is mandatory for meeting the precise timing requirements for the procedures.

Organizing a dialog through the time-sensitive network and checking it by digital signatures. The testing with digital signatures is organized in three levels: testing of the network data streams, testing of the timing characteristics, and testing of the interface with sensors and actuators. The network dialog is treated as a binary data sequence. A signature of this data sequence is calculated and stored as a standard (etalon) signature. Periodically, new signature is calculated and compared to the standard. If they are not equal, network messaging has been violated by a software bug, data degradation, or improper behaviour of the sensors and actuators. The timing of the sent and received packets can also be tested by signature analysis. The timing characteristics are calculated in advance over the standard timing diagram and some signature is calculated. Background routines periodically execute the functions of each sensor or actuator interface and record the data. A signature is calculated for the collected data and compared with its standard signature.

The signature is a 64-bit polynomial calculated by Advanced Encryption Standard (AES) hashing [10]. During testing a new signature is calculated that is compared with the etalon signature. The probability to detect timing differences by such signature analysis is very high:

$$P = 100 - 100\frac{H(m-n)(2^{m-n}-1)}{2^m-1}\ \%, \qquad (1)$$

where m is the length of the sequences and $n=64$, H is a step function to zero if $m<n$. When m is very large, the possible error goes to 2^{-n}.

Estimating and minimizing the interference of the testing itself is a critical aspect, as testing may significantly modify system behaviour. We implement non-intrusive tests that are periodically activated in the tested IoT production cell. Non-intrusiveness of the test allows the IoT-device to operate normally in time as if the test did not run. Intrusiveness could be measured by estimating the percentage of the delay caused in device performance by the test itself. Assume τ is the time of the test that checks some operation α with execution time T_α. A quantitative estimation of the intrusiveness is $(\tau / T_\alpha)*100\%$, which is the percentage of the time consumed by the test itself. A test is classified as non-intrusive if the above ratio is so small that the test has no effect on the operations.

The test must not have any influence on the worst-case execution time of the operations. In hard real-time operation the performance can be strongly reduced when using intrusive testing methods. Our nonintrusive test is based on the popular technique of merging testing procedures with critical operations. Such procedures store test information into an array at run time. Later, the simulator analyses the information in the array in background mode.

3.3. XML stream simulation

XML stream technology is adopted to deliver instructions to the manufacturing IoT cells and to collect production results. Every embedded system can take only its encrypted section from the stream. The stream is considered a sequence of events without tree construction of the XML document.

The testing of this important function is based again on simulation. A simulated XML stream is bouncing among the IoT embedded systems. The section for every embedded system (Figure 2) starts with a non-encrypted identifier of the embedded system. In the local network, every embedded system reads the XML stream and, by comparing the identifiers, reads or not the body, which itself is encrypted by symmetric cryptography.

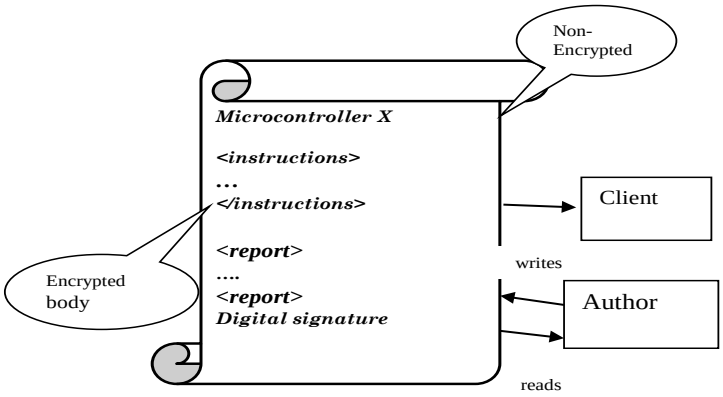

Fig.2. A section of the simulating XML stream

The integrity and the authenticity of the section are performed again by digital signatures. A thread parses the section and, by taking the role of the author, inserts into the stream section a report, which contains the results of the testing for some period. At the end of the report, there is a general assessment of the stable state of the manufacturing cell. The Internet software stack must respond adequately to the injected XML stream, reading its designated section and responding within the report. If some embedded system does not perform correctly, it is declared in non-stable state. The simulated XML stream also checks the interoperability of the IoT embedded systems, assessing the compliance of the Internet functions with the standard specification (recommended reading is [11]).

The manufacturing lines are upgraded often with new computerized equipment that must be checked for the new role in the network environment regard the timing and communication interoperability standards. Our simulator runs according to the standard specifications of the capabilities. It generates stimuli to the target devices and compares the correctness of the messaging sequence. The test cases (stimuli and responses) are parametrized with some tolerances that are within the range of correct behaviour. The interoperability of IoT embedded systems also requires supporting

web-based testing. The simulator runs in one of the IoT systems and generates the following sequence to every other system under test: initialization by stimuli to invoke a specific state; validation of the communication; capturing content of some of the packets; verification and concluding assessment of the desired functionality.

3.4. Web Services as Transactions

The manufacturing IoT embedded systems use web services to exchange messages with other embedded systems as well as with CAD and control data centres. Web services are remote procedure calls based on XML vocabularies (SOAP gives format of the messages; WSDL describes the data and operations in a web service).

For test results collection, one of the embedded systems could be designated as a 'master' to periodically collect data about the state of the others. Below is an example of a web-service request for the invocation of a self-test:

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope>
  <env:Body>
    <nso:runStableTest>
     <dateTime xsi:type="xsd:date">09/20/2019</dateTime>
     <mode xsi:type="xsd:string">Mode</mode>
    </runStableTest>
  </env:Body>
</env:Envelope>
```

The request operation is *runStableTest.* The operation attributes are the date/time and the mode of the self-test. The self-test initializes the software of the embedded system and invokes procedure that activates a dummy run of the peripheral devices and of the interrupt handlers for every device. At the end, the system must be re-initialized. The response can look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope>
  <env:Body>
    <runStableTestResponse>
     <xmlOut xsi:type="xsd:string">…
       ….
      <testTime>12:03:35</testTime>
       <stateTable>……</stateTable>
    </runStableTestResponse>
  </env:Body>
</env:Envelope>
```

The execution of the web service test has to be atomic. It cannot be pre-empted in the middle by some system error, which can put the IoT cell in an unpredictable and/or unreliable state. For that reason, we wrap all operations that are invoked by web services as transactions. This assures that all operations are encapsulated entirely in a single logical unit of work and are executed in an all-or-nothing mode. Transactions can commit (fully successful) or abort (transaction failed). Aborted transactions must be rolled back to undo any changes they performed.

The rollback procedure is organized by a log array of the initial state before the execution of the self-test operations. It records all modifications in a sequence of pairs: "… operation *i* starts – operation *i* commits …". If there is no indication of commit, the initial state before the operation is restored. This is especially important for the self-test that performs dummy execution of all interrupt handlers. Aborted transactions must leave all handlers in non-active state.

3.5. IoT Embedded System Emergency Reboot

Manufacturing environments are noisy, and some errors are difficult to discover and differentiate from the normal conditions. They are often sporadic and are caused by the unstable mode of the sensors and actuators. In our development, we implement rebooting for recovering the initial state.

We organize a partial reboot by deploying some properties of the development environment, applying them only to the interrupt handlers that manipulate the machinery sensors and actuators. When some stuck-at-unknown state conditions occur, some coarse monitoring of each individual interrupt handler takes place, prompting a restart.

Handlers, which work with sensors and actuators, implement a procedure that allows rebooting of the external devices into a well-known state. The procedure implements the policy that considers a thread unavailable unless it replies on time. Handlers analyse each interrupt and possible related errors and, depending on the nature of the error, may force the embedded system to ignore them or reboot.

4. DISCUSSION AND FUTURE WORK

In this section, we comment some of the contributions of our development.

The design of self-testing suite is based on functional simulation methods. Simulation [12] is a classical technique to study static or dynamic time-dependent systems in which delay is important property. In our design, simulation is a collection of threads that study by stimulus-responses the dynamic characteristics of the system and characterize its current state. The simulating threads are not incorporated in the operating system, but they run on application level as user threads and in some cases as some dummy interrupt handlers. Every IoT embedded system provides entry vector for every possible interrupt from a peripheral device. Such peripheral devices are several dozen or more, but the real systems use in average 50% of them. Our

simulation threads replace the empty device handlers. Such implementation brings some risks because the incorporated simulation threads could have bugs. To reduce the risk, we keep them very short and simple. The “interrupts” to the simulating thread are generated by software interrupt instructions from a simple monitor that works in cooperation with the central interrupt controller. Simulating threads are triggered when the application programs start initialization of some utility service or some peripheral devices.

The testing of the main and timing characteristics of the time-sensitive network of IoT embedded systems is organized by digital signatures that are incorporated into the simulator. The embedded test routines perform non-intrusive testing considering the worst-case execution time.

Using digital signatures for timing analysis cannot be considered intrusive testing for two reasons: a. some of the test subroutines use polynomial way to present the count of the clock cycles using the system timer; b. etalon signatures in some cases are calculated in background mode.

Check of the common time of all devices including the switches of the local network depends on the specifics of the incorporated microcontrollers. The most practical approach is to incorporate such check in the handler of the Reset interrupt.

Web services, as the main technology for Internet messaging, are tested by bouncing services between the cells. Such testing web services are simple, and they are considered as commit-or-rollback transactions during testing. The commit time is limited and if there is a timeout, the web service has to rollback. In timeout case the simulator thread starts to check the stable state of the system.

In general, the implementation of such self-testing approach is a challenge. The simulator threads run in very dynamic multithreaded environment where the operating system has little control over the synchronization between the threads. The interrupt priorities are changed accordingly. The simulator threads have lowest priority by default to prevent any unexpected influence.

As some future development, the presented self-testing methods could be combined with chaos engineering that randomly causes breakdowns in the system. Such “chaotic” intervention can prove both the steadiness of the manufacturing cell and increase the confidence in its capability [13, 14].

In manufacturing preferable could be the following types of breakdowns:

a. Some delays or other timing disruption (time attack) could be inserted and then some check of the stable state of the system could be performed;
b. In the manufacturing network some cell could be declared not-responding and check how the CAD or control centre could respond;
c. Some web service could contain wrong specification of the machinery, after that some analysis can be made of the reaction of the IoT system.

5. CONCLUSION

In this publication, we have shared our design methods for testing of an IoT-enabled system for manufacturing. Most of the described approaches are based on our professional experience; others are at the level of concept definition and experiments. Contributions of the paper are in the area of continuing development of the architecture of advanced manufacturing. The stable (steady) state of the entire manufacturing complex is of dominating importance. We hope that with the continued advances in self-testing by simulation the IoT-enabled production automation can help materialization into fully autonomous systems and can provide services in the future to cloud manufacturing.

REFERENCES

[1] IoT standard and protocols, 2019 comparisons of networks, protocols and standards, *https://www.postscapes.com/internet-of-things-protocols/*

[2] Calvaresi, D., Marinoni, M., Sturm, A., Schumacher, M., The challenge of real-time multi-agent systems for enabling IoT and CPS, *WI '17- Proceedings of the International Conference on Web Intelligence*, August 23-26, Leipzig, Germany, 2017.

[3] Automation ML, *https://www.automationml.org/o.red.c/home.html*

[4] XML Representation of STEP Schemas and Data, *xml-representation-step-schemas-and-data*

[5] Freeman, E., Gelernter, D., Document stream operating system, *US Patent* App. 11/607,099, 2006.

[6] Lemos A.L., Florian D., and Boualem B. Web service composition: a survey of techniques and tools, *ACM Computing Surveys (CSUR)* 48.3, 33, 2016.

[7] Kim H., Ahmad A., Hwang J., Baqa H., Gall F.L., Ortega M.A.R., Song J., IoT-TaaS: Toward a Prospective IoT Testing Framework, *IEEE Access Journal*, vol. 6, April 2018, pp. 15480-15493.

[8] Van de Velde G. *et al, RFC 4864, Local Network Protection for IPv6*, May 2007.

[9] IEEE 802.1Qbv. Enhancements for Scheduled Traffic, *www.IEEE802.org*, 2016.

[10] Daemen J., Rijmen V., AES Proposal: Rijndael (PDF), *National Institute of Standards and Technology*, Archived (PDF) from the original on 5 March 2013.

[11] P. Rosenkranz, M. Wählisch, E. Baccelli, L. Ortmann, A Distributed Test System Architecture for Open-source IoT Software, *Proceeding IoT-Sys '15 Workshop on IoT challenges in Mobile and Industrial Systems*, p.p. 43-48, Florence, Italy, May 18 - 18, 2015.

[12] Sokolowski, J.A., Banks, C.M., Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains, *John Wiley & Sons*, 2010.

[13] Orzell, G., Izrailevsky, Y., Validating the resiliency of networked applications, *US Patent US20120072571 A1*.

[14] Principles of Chaos Engineering, *principlesofchaos.org*. Retrieved 2017.

Information about the authors:

Iliya Georgiev – Department of Mathematics and Computer Sciences, Metro State University of Denver, USA, Professor

Ivo Georgiev – Department of Mathematics and Computer Sciences, Metro State University of Denver, USA, Adjacent Professor

Manuscript received on 2 July 2019