Experiences with Service-Oriented Middleware for Dynamic Instrumentation of Enterprise Distributed Real-time and Embedded Systems [⋆]

James H. Hill[1,⋆⋆], Hunt Sutherland[4], Douglas C. Schmidt[2], Thomas Silveria[3], John M. Slaby[3], Paul Staudinger[4], and Nikita A. Visnevski[4]

[1] Indiana U./Purdue U. at Indianapolis, 723 W. Michigan Street, Indianapolis, IN 46202
[2] Vanderbilt University, VU Station B #351679, 2301 Vanderbilt Place, Nashville, TN 37235
[3] Raytheon Company, 1847 W Main Rd, Portsmouth, RI 02871
[4] GE Global Research, 1 Research Circle, Niskayuna, NY 12309

Abstract. Test and evaluation (T&E) of enterprise distributed real-time and embedded (DRE) system quality-of-service (QoS) during early phases of the software lifecycle helps increase confidence levels that the system under development will meet its QoS requirements. Conventional T&E techniques are complex since they tightly couple system implementations with metrics of interest. To help alleviate this complexity, the Embedded Instrumentation Systems Architecture (EISA) initiative defines an architecture that provides a metadata-driven methodology for heterogeneous data collection and aggregation in a synchronized and time-correlated fashion.
This paper describes our experiences applying an EISA-based T&E middleware framework to the Unified SHIP platform, which is a representative system for next-generation shipboard computing systems. The middleware framework discussed in this paper enables instrumenting shipboard computing systems to collect and extract metrics without *a priori* knowledge of the metrics collected. We found that the flexibility of EISA's metadata-driven approach to instrumentation and data collection increased developer and tester knowledge and analytical capabilities of end-to-end QoS in shipboard computing systems.

1 Introduction

Challenges of developing shipboard computing systems. Shipboard computing systems are a class of enterprise distributed real-time and embedded (DRE) systems with stringent quality-of-service (QoS) requirements (such as latency, response time, and scalability) that must be meet in addition to their functional requirements [9]. To ensure QoS requirements of such systems, system developers must analyze and optimize end-to-end performance throughout the software lifecycle. Ideally, this test and evaluation (T&E) [2] process should start in the architectural design phase of shipboard computing, as opposed to waiting until final system integration later in the lifecycle.

⋆ This work was sponsored in part by The Embedded Instrumentation (EI) Group of the Test and Evaluation/Science and Technology (T&E/S&T) Program Office of the Test Resource Management Center (TRMC).
⋆⋆ Contact arthor's email address: `hillj@cs.iupui.edu`

T&E of shipboard computing system QoS requirements typically employs software instrumentation techniques [1, 5, 7, 9] that collect metrics of interest (*e.g.*, CPU utilization, memory usage, response of received events, and heartbeat of an application) while the system executes in its target environment. Performance analysis tools then evaluate the collected metrics and inform system developers and testers whether the system meets its QoS requirements. These tools can also identify bottlenecks in system and application components that exhibit high and/or unpredictable resource usage [3, 6].

Although software instrumentation facilitates T&E of shipboard computing system QoS requirements, conventional techniques for collecting metrics are tightly coupled to the system's implementation [2, 9, 11]. For example, shipboard computing developers often decide during the system design phase what metrics to collect for T&E, as shown in Figure 1. Developers then incorporate into the system's design the necessary probes

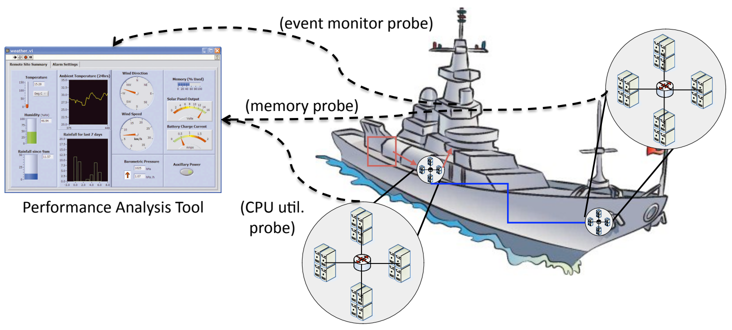

Fig. 1. Conventional Way to Instrument Shipboard Computing Systems

to collect these metrics from the distributed environment.

The drawback with a tightly-coupled approach is that shipboard computing developers must either (1) redesign the system to incorporate the new/different metrics or (2) use *ad hoc* techniques, such as augmenting existing code with the necessary interfaces without understanding its impact to the overall system's design and maintainability, to collect such metrics. Developers therefore need better techniques to simplify instrumenting shipboard computing systems for collecting and extracting metrics—especially when the desired metrics are not known *a priori*.

Our approach → EISA-based T&E framework. The Embedded Instrumentation Systems Architecture (EISA) [10] initiative defines a metadata-driven method for heterogeneous data collection and aggregation in a synchronized and time-correlated fashion [10]. EISA uses a data-centric approach to instrumentation and data collection for T&E, as opposed to an interface-centric approach. Instead of integrating into the system's design many interfaces and methods to extract and collect metrics, EISA treats all metrics as a arbitrary data that flows over a common reusable channel and discoverable via metametrics[5] [8], as shown in Figure 2. EISA thus helps reduce the coupling

[5] Metametrics are metadata that describe metrics collected at runtime without knowing its structure and quantity *a priori*.

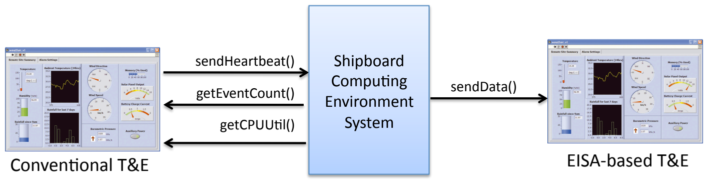

Fig. 2. Conventional Approach vs. EISA's Approach to T&E

between system design and instrumentation logic incurred with the conventional T&E techniques described above.

This experience report discusses our insights and lessons learned while developing and applying the *Open-source Architecture for Software Instrumentation of Systems (OASIS)* in the context of shipboard computing systems. OASIS is a service-oriented middleware framework that enables lightweight dynamic instrumentation of EISA-based T&E from the domain of shipboard computing. OASIS provides techniques and tools that enable shipboard computing developers and testers to (1) collect metrics from a distributed environment at runtime without *a priori* knowledge of what metrics are being collected, (2) apply performance analysis tools to evaluate QoS without being tightly coupled to the system's implementation, and (3) use the results of the metric analysis to evaluate end-to-end QoS of application and infrastructure components throughout their lifecycle.

Our experiences gained from developing and applying OASIS to shipboard computing systems show that EISA's metadata-driven approach to instrumentation and data collection provides flexibility that can increase DRE system developers and tester's knowledge base and analytical capabilities of end-to-end QoS. We also found that OASIS helped quantify which technologies were most beneficial for collecting metametrics.

Paper organization. The remainder of this paper is organized as follows: Section 2 summarizes the challenges that motivate the need for OASIS in the context of shipboard computing; Section 3 describes how OASIS addresses these challenges; and Section 4 presents concluding remarks and lessons learned.

2 The Unified SHIP Platform

In previous work [10], EISA-based tools were used to instrument hardware components (*e.g.*, sensor hardware components) of enterprise DRE systems. These systems, however, are composed of both hardware and software components. Ideally, end-to-end QoS evaluation of shipboard computing systems should employ performance analysis of both hardware and software components.

To help evaluate EISA in a representative enterprise DRE system, we created the *Unified Software/Hardware Instrumentation Proof-of-concept (Unified SHIP)* platform, which provides a representative environment for investigating technical challenges of next-generation domain of shipboard computing systems. As shown in Figure 3, the Unified SHIP platform contains software components (*i.e.*, the rectangles in Figure 3) implemented using the Component Integrated ACE ORB (`www.dre.vanderbilt.edu/CIAO`), which is a C++ implementation of the Lightweight CORBA Component Model [4]. Conversely, performance analysis tools are implemented using a variety of programming languages, such as C++, C#, and Java. The software applications run on real-time Linux and Solaris operating systems, whereas performance analysis tools run on Windows and conventional Linux operating systems.

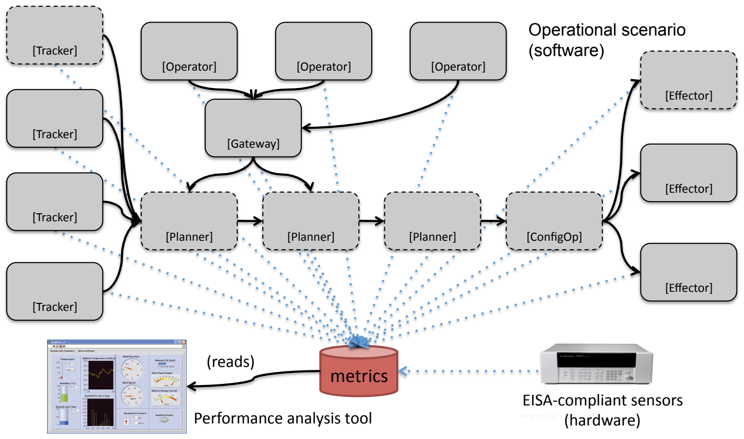

Fig. 3. Overview of the Unified SHIP Platform

Figure 3 also shows how the Unified SHIP platform consists of EISA-compliant sensor hardware components and a collection of software components that performed the following operational capabilities for shipboard computing systems: 4 components are trackers that monitor events in the operational environment, 3 components are planners that process data from the sensor components, 1 component performs configuration of the effectors, 3 components are effectors that react to commands from the configuration component, 3 components allow operators to send commands to the planner components, and 1 component is a gateway that authenticates login credentials from the operator components. The directed line between each component in Figure 3 represents inter-component communication, such as sending an event between two different components.

Existing techniques for instrumenting shipboard computing systems assume software instrumentation concerns (*e.g.*, what metrics to collect and how to extract metrics from the system) are incorporated into the system's design. Since the Unified SHIP platform consists of hardware and software components at various degrees of maturity and deployment, it is hard to use existing instrumentation techniques to collect and extract metrics for QoS evaluation during early phases of the software lifecycle. In particular, developers and testers of the Unified SHIP platform face the following challenges:

• **Challenge 1: Describing metametrics in a platform- and language- independent manner.** The heterogeity of the unified SHIP platform's software and hardware components makes it undesirable to tightly couple performance analysis tools to the target platform and language of software and hardware components to collect and analyze metrics. Platform- and language-independent techniques and tools are therefore needed that will enable description of metrics collected from hardware and software components.

• **Challenge 2: Collecting metrics without *a priori* knowledge of its structure and quantity.** Metrics collected via instrumentation in the Unified SHIP platform come from heterogenous sources, which make it tedious and error-prone for system developers and testers to tightly couple the systems implementation to understand each metric and technology *a priori*. Techniques are therefore needed that will enable the collection of metrics from the Unified SHIP platform for QoS evaluation without *a priori* knowledge of which metrics are collected.

The remainder of this experience report discusses the OASIS techniques and tools we used to address these challenges in the context of the Unified SHIP platform.

3 Applying EISA-based T&E to the Unified SHIP Platform

This section discusses the structure and functionality of the *Open-source Architecture for Software Instrumentation of Systems* (OASIS), which is service-oriented middleware that enables lightweight dynamic instrumentation of enterprise DRE systems. We also describe how OASIS was applied to address the Unified SHIP platform challenges identified in Section 2.

3.1 Overview of OASIS

The Unified SHIP platform discussed in Section 2 introduced several challenges that system developers and testers encounter when instrumenting shipboard computing systems to collect and extract metrics for performance analysis tools. These challenges involve describing, collecting, extracting, and analyzing metrics without *a priori* knowledge of the structure and quantity of metrics with respect to the underlying middleware and system infrastructure.

To address these challenges, we have developed OASIS to collect and extract metrics of interest without *a priori* of their structure or quantity. Metric collection and extraction in OASIS is independent of specific technologies and programming languages, which decouples OASIS from shipboard computing software details, such as incorporating into the system's design the necessary probes to collect and extract metrics for performance analysis. System developers and testers are thus not constrained to make decisions regarding what metrics to collect for performance analysis tools during earlier phases of the lifecycle.

Figure 4 presents an high-level overview of OASIS. As shown in this figure, OASIS consists of the following five entities:

• **Software probes**, which are autonomous agents responsible for collecting metrics of interest in the system, including the current value(s) of an event, the current state of a component, or the heartbeat the component or the node hosting the component. Software probes are considered autonomous agents since they act independently of OASIS. For example, an event monitor software probe from the Unified SHIP platform may

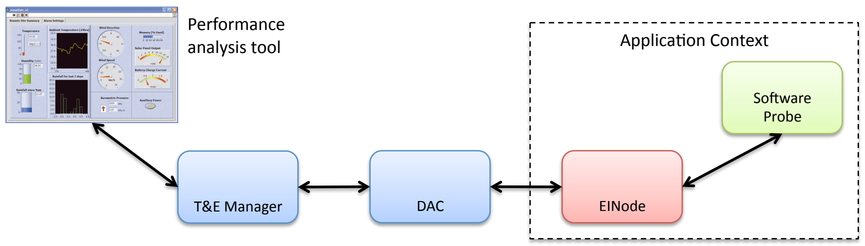

Fig. 4. High-level overview of the OASIS middleware.

send metrics every time a component receives an events; where as, a heartbeat software probe may send metrics at a periodic rate. Finally, each software probe is distinguished by an user-defined UUID and corresponding human-readable name.

There are two types of software probes in OASIS: *application-level probes* and *infrastructure-level probes*. Application-level probes are embedded into application components to collect metrics of interest, such as the state of a component or number of events sent/received. Infrastructure-level probes collect metrics that are not easily available at the application-level or may collect redundant metrics at the application-level, such as current memory usage or heartbeat of each host in the target environment. Both application- and infrastructure-level probes submit their metrics to the embedded instrumentation node described next.

• **Embedded instrumentation node (EINode)**, which is responsible for receiving metrics from software probes. OASIS has one EINode per application-context, which is a domain of commonly related data. Examples of an application-context include a single component, an executable, or a single host in the target environment. The application-context for an EINode, however, is locality constrained to ensure data transmission from a software probe to an EINode need not cross network boundaries, only process boundaries. Moreover, the EINode controls the flow of data it receives from software probes and submits to the data and acquisition controller described next. Each EINode is also distinguished by a unique user-defined UUID and corresponding human-readable name.

• **Data acquisition and controller (DAC)**, which receives data from an EINode and archives it for acquisition by performance analysis tools, such as querying the performance of the latest state of component collected by a application-level software probe. The DAC is a persistent database with a consistent location in the target environment that can be located via a naming service. This design decouples an EINode from a DAC and enables an EINode to dynamically discover at creation time which DAC it will submit data. Moreover, if a DAC fails during at runtime the EINode can (re)discover a new DAC to submit data. The DAC registers itself when the test and evaluation manager (see below) when it is created and is identifiable by a unique user-defined UUID and corresponding human-readable name.

• **Test and evaluation manager (T&E)**, which is the main entry point for user applications (see below) into OASIS. The T&E manager gathers data from each DAC that has registered with it. The T&E manager also enables user applications to send signals to each software probe in the system at runtime to alter its behavior, *e.g.*, by decreasing/increasing the hertz of the heartbeat software probe in the Unified SHIP platform scenario. This dynamic behavior is possible because the T&E manager is aware of all its DACs in the system, the DACs are aware of all its EINodes, and the EINodes are aware of all their registered software probes.

• **Performance analysis tools**, which are domain-specific tools, such as distributed resource managers and real-time monitoring and display consoles from the Unified SHIP platform, that interact with OASIS by requesting metrics collected from different software probes via the T&E manager. Tools can also send signals/commands to software probes to alter their behavior at runtime. This design enables system developers and testers and performance analysis tools to control the effects of software instrumentation at runtime and minimize the affects on overall system performance.

Figure 5 shows the integration of OASIS with the Unified SHIP platform. Each

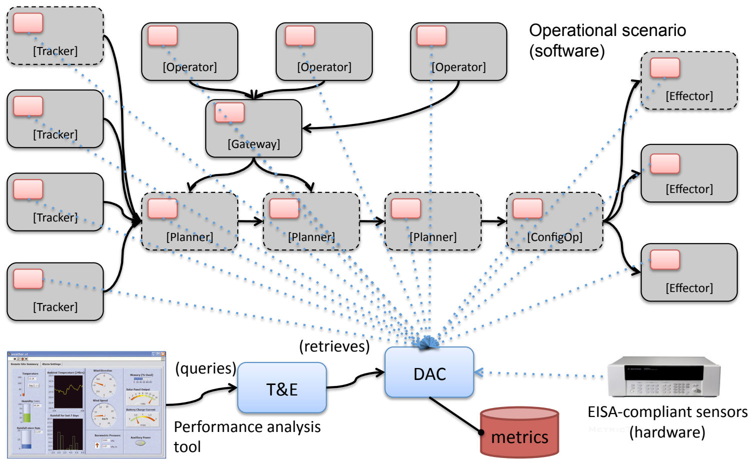

Fig. 5. Integration of OASIS with the Unified SHIP Platform

hardware and software component is associated with an EINode that contains a set of software probes (or instruments in the case of hardware components [8]) that collect and submit metrics for extraction from the system. When an EINode receives metrics from a software probe (or instrument), it sends it to a DAC for storage and on-demand retrieval. Performance analysis tools then request collected metrics via the T&E manager, which locates the appropriate metrics in a DAC.

3.2 Resolving T&E Challenges in the Unified SHIP Platform

The remainder of this section describes how we applied OASIS to address the two challenges presented in Section 2.

Solution 1. Using XML-based Techniques to Capture Metametrics. Challenge 1 in Section 2 pertained to system developers and testers of the Unified SHIP platform needing a technique for capturing platform- and language-independent metametrics. OASIS addresses this challenge by using XSL Schema Definition (XSD) to describe the metametrics in the Unified SHIP platform. XSD provides fine-grained description capabilities of data types (such as quantity and constraints) as opposed to other techniques (such as Interface Definition Language (IDL)) that only capture data types and structure).

```
<?xml version='1.0' ?>
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'
            xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
            xsi:schemaLocation='http://www.w3.org/2001/XMLSchema XMLSchema.xsd'
            elementFormDefault='qualified'
            version='1.0'>
  <xsd:element name='probeMetadata' type='component.state' />
  <xsd:complexType name='component.state'>
    <xsd:annotation id='metadata'>
      <xsd:appinfo>0A499B6B-7250-4B88-B9DC-360D32639081</xsd:appinfo>
      <xsd:documentation>Monitors a component's state</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name='component' type='xsd:string' />
      <xsd:element name='state' type='xsd:integer' />
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Listing 1.1. Example XSD File for Describing Metrics Collected by a software Probe

Listing 1.1 shows an example XSD file that describes metrics collected by a software probe for tracking a software component's state from the Unified SHIP platform. This listing shows the software probe has an UUID that identifies its metametrics. Each element in the XSD file represents a data point collected in the metrics, such as the name of the component as listed in Listing 1.1. This information is sent to the DAC at registration time (*i.e.*, before the Unified SHIP system is active). Performance analysis tools, such as distributed resource managers and real-time monitoring and consoles, then use the XSD file to learn about metrics of interest collected without requiring *a priori* knowledge how the metric was collected.

Solution 2. Using Binary Data and Sockets to Transmit Metrics. Challenge 2 in Section 2 pertained to collecting metrics of interest from a system without *a priori* knowledge of its structure and quantity. OASIS addresses this challenge by using binary data to represent metrics collected by a software probe and transmitting the metrics using traditional socket programming. It uses binary data and socket programming so it is not bound to a specific technology or programming language. Moreover, transmitting binary data over sockets significantly enhances performance (*e.g.*, response-time and latency) compared with using text-based formats, such as XML. Finally, prior work [10] has shown the easy of using socket programming to adapt hardware components to the EISA specification.

Figure 6 shows how metrics are collected from the Unified SHIP platform using OASIS. This figure shows how software probes package metrics from the Unified SHIP

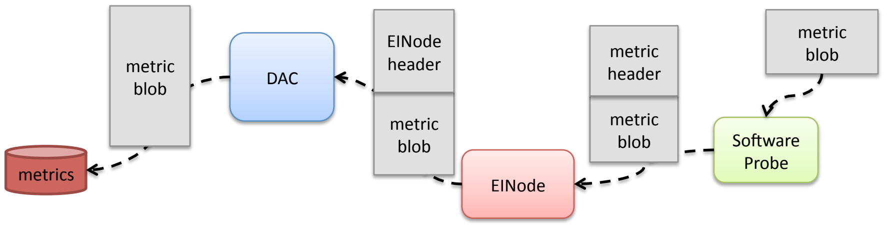

Fig. 6. Collecting Metrics as Binary Data in OASIS

platform into blobs of binary data. Before a software probe sends the metrics to an EINode, it prepends a metric header to identify the origins of the metric. After the EINode receives a packaged metrics, it prepends a location header that identifies what EINode is transmitting data and then sends the final package to the Data Acquisition and Controller for storage. Performance analysis tools (such as distributed resource managers and real-time monitors) then request collected metrics using the T&E manager and use metametrics (see Section 3.2) to analyze metric contents.

4 Concluding Remarks

Test and evaluation (T&E) of shipboard computing system QoS during early phases of the software lifecycle helps increase confidence that the system being developed will meet it requirements. Conventional T&E instrumentation mechanisms, however, are tightly coupled with the system's design and implementation. This experience report describes how the OASIS implementation of the *Embedded Instrumentation Systems Architecture* (EISA) initiative helped reduce these coupling concerns by applying a *metadata-driven* (as opposed to *interface-driven*) approach to T&E instrumentation continuously throughout the software lifecycle. OASIS enabled the DRE system to evolve throughout the software lifecycle without negatively impacting T&E instrumentation needs.

The following are lessons learned based on our experience thus far designing and implementing OASIS in the context of the Unified SHIP scenario described in Section 2:

• **Using a metadata-driven approach to shipboard computing system instrumentation provides a more flexible solution than using an interface-driven approach.** The flexibility of OASIS's EISA-based metadata-driven approach to instrumentation and data collection helped increase developer and tester knowledge and analytical capabilities of end-to-end QoS. In particular, OASIS's metadata-driven instrumentation of the Unified SHIP platform enabled our team to collect metrics from heterogeneous environments and technologies and develop analytical tools that were decoupled from metric sources or technologies.

• **XML is best used for describing metrics, not transmitting them.** We learned by applying OASIS to the Unified SHIP platform that XML is better used for describing metrics and metric metadata than transmitting metrics at runtime to the OASIS Data

Acquisition and Controller. In particular, the stringent QoS requirements of shipboard computing systems conflicted with the overhead incurred from transmitted XML-based metrics. Instead, we found it was more efficient to transmit metametrics using XML-based documents (such as XML Schema Definition) and use binary streams to transmit the actual metrics.

• **Although metadata is not transmitted at runtime, it must still be transmitted efficiently instead of haphazardly.** During registration time, an OASIS EINode transmits XML-based metadata that describes the metrics collected by its software probes. In some cases, an EINode may have different configurations of the same software probe, such as one event monitor probe that collects data on every event and another probe that collects data on every other event. For this case, the same metric metadata will be transmitted twice. Our future work is investigating techniques for optimizing transmission of metadata to reduce redundancy and unnecessary transmission.

OASIS has been integrated in the CUTS system execution modeling tool and is freely available for download in open-source from the following location: `www.dre.vanderbilt.edu/CUTS`.

References

1. B. Cantrill, M. W. Shapiro, and A. H. Leventhal. Dynamic instrumentation of production systems. In *Proceedings of the General Track: 2004 USENIX Annual Technical Conference*, pages 15–28, June 2004.
2. G. Hudgins, K. Poch, and J. Secondine. The Test and Training Enabling Architecture (TENA) Enabling Technology For The Joint Mission Environment Test Capability (JMETC) and Other Emerging Range Systems. In *Proceeding of U.S. Air Force T&E Days*, 2009.
3. D. A. Menasce, L. W. Dowdy, and V. A. F. Almeida. *Performance by Design: Computer Capacity Planning By Example*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.
4. Object Management Group. *Light Weight CORBA Component Model Revised Submission*, OMG Document realtime/03-05-05 edition, May 2003.
5. K. O'Hair. The JVMPI Transition to JVMTI. `java.sun.com/developer/technicalArticles/Programming/jvmpitransition`, 2006.
6. C. Smith and L. Williams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley Professional, Boston, MA, USA, September 2001.
7. A. Srivastava and A. Eustace. ATOM: A System for Building Customized Program Analysis Tools. In *PLDI '94: Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation*, pages 196–205, 1994.
8. A. Stefani and M. N. Xenos. Meta-metric Evaluation of E-Commerce-related Metrics. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 233:59–72, 2009.
9. Z. Tan, W. Leal, and L. Welch. Verification of Instrumentation Techniques for Resource Management of Real-time Systems. *J. Syst. Softw.*, 80(7):1015–1022, 2007.
10. N. Visnevski. Embedded Instrumentation Systems Architecture. In *Proceedings of IEEE International Instrumentation and Measurement Technology Conference*, May 2008.
11. D. G. Waddington, N. Roy, and D. C. Schmidt. Dynamic Analysis and Profiling of Multi-threaded Systems. In P. F. Tiako, editor, *Designing Software-Intensive Systems: Methods and Principles*. Idea Group, 2007.