Impact of Adding Security to Safety-Critical Real-Time Systems

A Case Study

Andrew J. Kornecki
College of Engineering
Embry Riddle Aeronautical University
Daytona Beach, Florida, USA
Andrew.Kornecki@erau.edu

Wendy F. Stevenson
College of Engineering
Embry Riddle Aeronautical University
Daytona Beach, Florida, USA
Wendy.Stevenson@garmin.com

***Abstract* — With the proliferation of complex interconnected embedded systems there is a need to ensure a level of security for those systems such that end-users will trust them to perform their required functions safely and securely. Incorporating security into a system invariably decreases the performance of the system. This is an important point to consider when designing embedded systems where the timing, space and power constraints are generally more stringent than in other systems. The presented case study proposes architecture for a secure interconnectivity component and evaluates impact of the security on the system performance. The paper suggests how informed decisions regarding tradeoffs between performance and security may be made.**

Keywords: Embedded Systems; Security; Safety; Modelling; Software Components; Performance; AADL

I. INTRODUCTION

Industrial control systems were traditionally isolated from the outside world which meant that malicious attacks could only be attempted at the site. This security threat was easily managed by controlling access to the premises. The ability to connect these control systems via a network cable meant that more functionality could be added to these systems. Examples of such functionality include remote monitoring, control, and data collection. Now control systems often have wireless transmitters and receivers built into them. This makes the systems vulnerable to remote security intrusions. While making the system more flexible and allowing complex applications to be developed, such modern control systems tend to increase the software assurance and security management efforts.

There are several levels of security that may be present in any given software controlled system. All applications require physical security management at the very least, even if such applications are not connected to any network. Security management must be intelligently applied to the system under development so as unnecessary effort is added. The supposed improvement to the system could be a liability to the system as a whole [1]. Additionally, the isolation of safety, security and system engineering practice may lead to a number of problems like inadequate understanding of safety and security semantics, incompleteness of requirements, and incompleteness of critical characteristics like verifiability and ambiguity in requirements [2].

Embedded systems have more stringent timing and memory usage constraints than conventional desktop systems. Therefore, the impact of adding security features and related tradeoff between the level of security and the associated performance and memory costs to the system must be carefully made.

All characteristics of a system are inter-related. Increasing the security of a system by adding more levels of authentication may have a negative impact on the performance of the system. For this reason, [3] suggest that the resulting architectural model be verified again using the currently available analysis and verification tools. Architecture Analysis and Design Language (AADL) promoted by the Society of Automotive Engineers since 2004 [4] is supporting the proposed analyses. AADL, with the help of security plug-ins, can validate whether or not a given architectural model violates some basic security principles [5]. The security policies must be defined at the architectural phase and any modifications to the policy at a later stage of development would likely require the architecture to be revisited.

Component-based software development has many advantages. After the interfaces have been well defined, components are designed, implemented, and verified independently which allows components to be easily modified without affecting the rest of the system. Components can be reused. However, when system is required to meet certain security requirements there is mistrust of using components of uncertain pedigree and thus reuse is rarely accepted option [6].

To meet the objectives of reusable components as well as increase the overall security of the system, the system is decomposed using the communicating processes pattern. As guidance, the granularity of the components should be small enough to enable each individual component to be formally proven correct (according to its specification) and yet large enough to enable the system to still meet its performance

requirements in terms of execution time as determined by AADL analyses.

In such framework the interconnectivity functionality may be implemented within a single component solely responsible for communicating over the network and with other systems. This component is the only component that needs to be designed to avoid, detect and respond to threats from the outside. If this component is secure, then it should not propagate intrusions to the other components in the system. So all the remaining components need only ensure that contain no software (or hardware) defects which pose a security risk to the system.

The Open Web Application Security Project (OWASP) maintains a list of known vulnerabilities specifically for web based applications. Some of these vulnerabilities relevant to embedded systems (adapted from [7]) are: injection, broken authentication/session management, insecure object references or cryptographic storage, miss-configuration, insufficient transport layer protection, and other vulnerabilities (code, error handling, general logic, and input validation).

Mitigation techniques include authentication, encryption and other means designed to avoid and/or detect intrusions. There is no method to evaluate the security of a new mitigation technique before it is used in a system. This is partly due to the fact that security is relative to the currently known possible attacks. What we can say is that a system is secure against currently known attacks.

The paper is organized as follows. After selection of the case study we introduce the architectural model of the system including the security component, discuss the assumption and then analyze the system using AADL and queuing models. The results and conclusions allow us to judge the viability of the proposed approach.

II. Selection of Case Study

The presented case study proposes architecture for a secure interconnectivity component and evaluates impact of the security on the system performance. The example selected is known architectural model for a Cooperative Adaptive Cruise Control System (CACC) [8,9] - an embedded control system for automobiles automatically monitoring and adjusting a vehicle's speed according to the traffic conditions in its immediate vicinity to maintain constant distance from the lead vehicle. The CACC receives the necessary data from sensors on the vehicle, communication with other CACC equipped vehicles (specifically the vehicle directly in front), and communication with a central Street Monitoring Data Center (SMDC). The system issues commands to the throttle and the brakes to effect a change in the vehicle's speed.

Changes have been made to the original CACC architectural model to enhance the level of security and analyses have been performed to assess the effectiveness of the changes. The reliability and availability of the enhanced model have been analyzed using a Markov model (not described in this paper which is focusing primarily on the impact of introducing security). Obviously, adding security functionality increases the execution time and the demand on the resources which is an important consideration for real time embedded systems where typically the resources are limited and response time requirements must be met. One needs to quantify the changes affected in the system by the addition of the interconnectivity component and other security functionality.

The model before adding the security functionality is compared to the security-enhanced model in terms of the processor(s) utilization.

AADL OSATE Tool Environment [10] with plug-ins allows the analyst to calculate flow latencies, analyze scheduling and security. AADL allows the analyst to model both hardware and software execution entities of the system. Software execution entities include processes, threads and data stores. Hardware execution entities include processors, busses and physical memory. Each entity has a set of properties associated with it: processor (clock frequency), process (period, scheduling and dispatch protocols), and thread (period, execution time, memory and processor connection binding). It is easy to modify the value of one or more properties and repeat the analysis. This allows different system configurations to be easily compared.

Two types of analysis can be done in AADL: scheduling and security analysis. The scheduling analysis attempts to determine the percentage utilization for each processor in the system based on the given (assumed) property values for each run time entity. The scheduling analysis answers the question, 'Given the processors with their properties and the threads with their properties, is there a configuration and schedule such that all threads meet their deadlines and execute at the required periods?' If such a configuration exists, the analysis calculates how busy the processors are likely to be, in the worst case.

A security level may be assigned to each execution entity and each data connection between the entities. Provided by the AADL security analysis checks that an entity and each connection has a security level which is greater than or equal to the security levels of all entities contained therein. The model proposed in the case study consists of just two security levels and thus such trivial analysis would be as follows. The highest security level is assigned to the interconnectivity component, and the lowest is assigned to the rest of the system entities. The interconnectivity component with high security level protects more vulnerable (i.e. less secure) parts of the system from possible threats and it is responsible for the defense against the threats. The resulting combined system is therefore more secure.

III. CACC Model

A. CACC Model Assumptions

In the CACC architectural model used in this case study, the distance between leading and following vehicles is a function of their speed difference, the condition of the road, the amount of traffic present and documented safe following distances. The users set the desired speed when there is no vehicle immediately in front of them. The long term trend is towards partial or even fully autonomous operation of a

single vehicle, or even groups of vehicles. Therefore, safety and security are essential attributes of CACC systems.

The devices interfacing with the CACC may be divided into two categories: (a) devices located on the vehicle and having a direct connection with the CACC, and (b) devices which communicate with the CACC via a wireless network connection (CACC systems on other vehicles and the SMDC monitoring the traffic in a given area).

It is assumed that attackers would not physically tamper with the vehicle itself in order to cause a breach in security. Possible breaches in security may come about through software defects in the user interface code which need to be discovered and corrected during the implementation phase. The case study focuses on the security issues visible at the architecture level of the system and so we will not be considering the user interface in more detail focusing instead attention on the external interface.

The CACC system communicates via wireless network connection to other vehicles in the vicinity and to a centralized Street Monitoring Data Center. These interfaces have the highest risk of malicious attack. Four major categories of attack are:

- Message Insertion - an invalid (made-up) SMDC/ CACC message is injected.
- Message Deletion - a valid SMDC/ CACC message is not received by the CACC system.
- Message Corruption - the contents of an SMDC/CACC message is altered before it is received.
- Message Flooding - multiple frequently repeated SMDC/CACC messages are received.

B. CACC Architectural Model

The execution view of an architectural model shows the execution (runtime) entities of the system as a screen shot captured from the OSATE tool (Fig. 1).

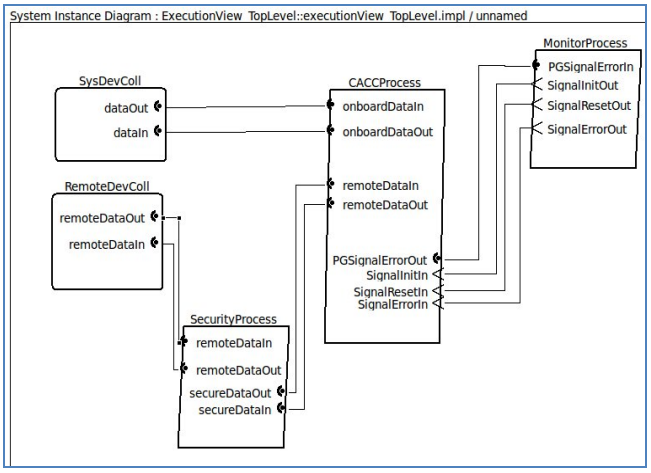

Figure 1. Security Enhanced CACC System

The boxes labeled *SysDevColl* and *RemoteDevColl* represent subsystems for the two types of devices. The remote devices, those that are connected via wireless network connection, are placed in a separate system to highlight the fact that the devices are not co-located on the vehicle. The connections between the devices and the *CACCProcess* are modeled using port groups.

CACCProcess and *MonitorProcess* are major functional processes of the system. Processes may be allocated to different processors provided that they are connected to a common bus. In this case study, each process was assigned a periodic scheduling algorithm with its own protected memory space and a group of threads. Hardware of the system includes two processors: an Intel processor with a frequency of 500 Mhz, and an FPGA processor with a frequency of 1.2 GHz. The *MonitorProcess* is bound to the Intel Processor and the *CACCProcess* is bound to the FPGA processor allowing for more effective system monitoring.

The original CACC model was modified to incorporate security. In the architectural model it is presented by adding an additional process dedicated to security aptly called *SecurityProcess*. The addition of the *SecurityProcess* requires minor modification of the *CACCProcess* interface: *remoteDataIn* and *remoteDataOut* data ports need to be connected to the *SecurityProcess* instead directly to the *RemoteDeviceColl* as in the case without security.

C. Security Process Architectural Analysis

Within the *SecurityProcess* there are two distinct functionalities that deserve their separate components as shown in Fig 2 captured from OSATE tool. First, there is a need to authenticate the sender of the messages. We introduce thus an *Authentication* component (thread group). The reason for separation is that the authentication functionality can be run on separate dedicated hardware or on a separate processor which makes the potential configurations more flexible. Secondly, this component can be developed, tested and verified in isolation of the rest of the system. If it is developed well it may even be reused for subsequent systems. This *Authentication* component interfaces to the rest of the system via event data ports.

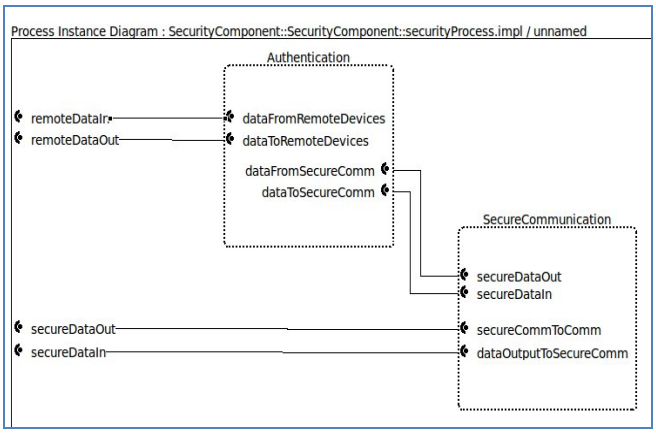

Figure 2. SecurityProcess Thread Groups.

Conceptually, there is a message queue for each possible input and output to the rest of the systems. An event is triggered whenever a message appears in the queue. Having messages in a queue structure allows messages to be validated before being sent to the intended recipient. In

addition, to mitigate the possibility of lost or injected messages, the system needs to monitor the time between each subsequent message of a particular type. This implies that specific inter-message delays should be defined in the system specification. The actual functionality which fulfills these requirements is implemented in the authentication component.

The *SecureCommunication* component looks very similar to the *Authentication* component. However, their respective functionalities differ. The authentication thread group is responsible for verifying the sender and the content of the messages while the secure communication thread group is responsible for passing the data in a secure way into the rest of the system.

The data is received via an event data port. This implies that the message is placed in a queue and the thread receives notification that a message has arrived. The thread can then process the message according to the message type. If an invalid message is encountered, the thread rejects the message, ensuring that the rest of the system is protected from erroneous messages.

The more intelligence that is built into this message verification the more secure the system will be since it will be able to detect more erroneous messages and stop them from entering the rest of the system. However, a trade-off needs to be made between the complexity of the thread and its worst case scenario execution time. The system still needs to be able to meet its timing requirements.

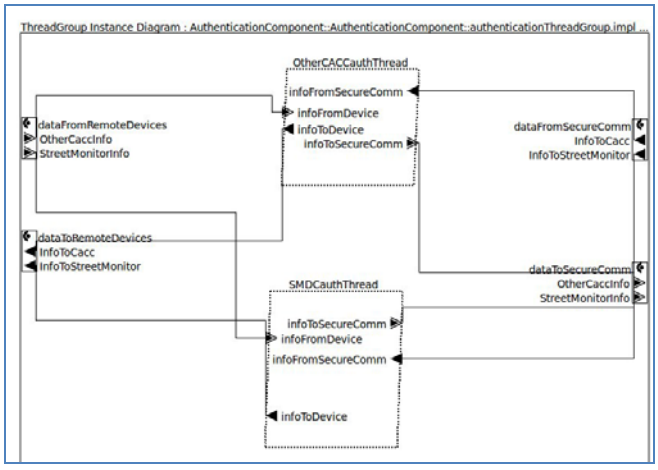

Figure 3. Authentication Thread Group.

From the software perspective the components represent groups of threads. The thread groups are further broken down in the OSATE tool to show more detail. As an example, the *Authentication* component is shown on Fig 3. Each data channel that is to be secured is handled by a separate threads dedicated to handling OtherCACC and SMDC data (*OtherCACCauthThread* and *SMDCauthThread* respectively). This is to ensure that the system will be more easily verified to meet the timing constraints for each channel. It also allows each thread to be scheduled according to the period in which we receive the messages. Similarly, *SecureCommunication* component consists also of two threads dedicated to OtherCACC and SMDC data.

IV. Data Analysis

A. Data Analysis Assumptions

To determine numerical values required for further analysis several assumptions were made. Timing of two authentication algorithms implemented in hardware and software was described in literature [11]. For an FPGA processor with a frequency of 1 GHz, the average time taken to authenticate a packet using hardware was 8µs for the size of the packets uniformly distributed between 40 and 1500 bytes. Using software, the average time was 27 µs. The example CACC system is using FPGA with a frequency of 1.2 GHz which implies that the execution time for the authentication will be lower than the above results. We are considering the worst case and therefore we can use this value as a worst case estimate for our system. Since this is the average estimate, we arbitrary add 50% to obtain an estimate for the worst case execution time for the algorithm (assuming a normal distribution).

In the original model, data from the other CACC systems and the SMDC were received once every 100 ms each. The security component threads were assigned two times higher activation rate deemed sufficient to receive all the messages (assuming that there is no flooding of messages). All CACC message types incoming from speed and gap sensors, GPS, brake, user interface, SMDC and other CACC equipped vehicles were analyzed to determine their content, size, frequency, and the inter-arrival rate. Except the user interface and the messages incoming from other CACCs, the messages are assumed to be periodic. The messages from the other CACCs are transmitted periodically. However, since there may be multiple vehicles in the surrounding area and the vehicles' clock times are not synchronized, these messages may be received in a random manner. Each vehicle broadcasts a message to other vehicles every 100ms. Since CACC is a single lane collision avoidance system, it is assumed that there are at most five vehicles in the surrounding area from which messages are received. The exponential distribution was used to model this scenario.

The worst case message processing time is the time taken, in the worst case, by the server to receive the message bytes, parse the message into an appropriate form and send it to another thread. The processing time is a function of the message size as well as the target processor.

For the purposes of this case study, it is assumed that the worst case message processing rate for the target system is 100 bytes per millisecond on the FPGA processor. Since we analyze the increase in utilization after adding security functionality, the actual value is not that critical. Once the worst case processing rate is applied to the estimated message sizes the worst case message processing times for each message type was determined.

Based on the above analyses, the frequency of each message type was defined and used to specify the period for each thread responsible for processing that message. In order to authenticate, verify and validate the messages from

the remotely connected devices, additional effort is required when these messages are received by the CACC System. To estimate the impact that this additional effort would have on the message processing functionality of the CACC System, we model this message processing using queuing theory.

The threads in the *SecureCommunication* thread group were assigned a period of 50 ms in keeping with the *Authentication* threads. The frequencies of the message arrivals were set to match the periods of the threads needed to process them, with the exception of the user interface and the other CACC. The period used for the user interface thread would depend on the responsiveness required. For the purpose of this case study, it is left at 200ms. It is more difficult to determine the period required for the other CACC thread. To make the scheduling of tasks more flexible, the frequency of the CACC thread is set to 100ms with the assumption that the thread is able to process up to five messages in one execution. In practice, it might be infeasible to process up to five messages within a single cycle and this period would then need to be reduced.

Each thread within each thread group was assigned both an execution time and a period. To obtain the execution times for the *SecurityProcess* threads the cycle times of the two processors were used. For example, the SMDC secure communication thread has an execution time of 0.4 ms taking 571,428 cycles on the FPGA processor with a cycle time of 700ps. The same thread on the Intel processor with a cycle time of 2μs, it would take 2* 571,428 μs = 1.143 ms.

TABLE I. ASSIGNED EXECUTION TIMES AND PERIODS FOR THE SECURITY ENHANCEMENT SYSTEM THREADS

Component	***Thread***	***Execution Time FPGA***	***Execution Time Intel***
Authentication	*SMDC*	0.041ms	0.117ms
Authentication	*OtherCACC*	0.041ms	0.117ms
SecureComm	*SMDC*	0.4ms	1.143ms
SecureComm	*OtherCACC*	0.4-2.0ms	1.143-5.71ms

B. AADL Scheduling Analysis

The above assumptions resulted in calculation of the approximate execution times for the *Authentication* and *SecureCommunication* thread group as assigned to individual threads (handling the *SMDC* and the *OtherCACC* data). The resulting numerical values, as shown in Table I, were used in subsequent analyses. All security related threads were assumed to run with period of 50 ms.

OSATE Scheduling analysis was used to determine the estimated percentage utilization of each processor in the system. This analysis uses information about scheduling algorithms assigned to each processor, the frequency of each processor, the period and the execution time of each thread.

The system has five modes of operation, namely, *Initialization, Disengaged, Engaged, Error,* and *Off*. The analysis was done using only the *Engaged* mode since the utilization of the processors will be highest in this mode. It is possible to perform this analysis for each mode of the system. For the original model the processor utilizations were 67% for Intel and 51% for the FPGA. With the *SecurityProcess* bounded to FPGA, the FPGA utilization grew to 55% (a 7.8% increase). The Intel processor utilization grew to 76% when *SecurityProcess* was bound to the Intel processor (a 13.4% increase).

Since the Intel processor is slower, the percentage increase is higher when binding the *SecurityProcess* to the Intel rather than to the FPGA processor. The security functionality would impact the performance of the system more if it is assigned to the Intel processor. However, there are advantages to be gained by having separate security functionality. It is easier to modify the security component, including the hardware. Perhaps more important is a separation of concerns within the system.

The component responsible for monitoring the safety of the system is bound to the Intel processor. If the security component is also bound to Intel processor, it means that all interfaces between the outside world and the main functionality of the system on the FPGA processor need to pass through the Intel processor. Thus, more effort may need to be expended in making the monitoring processor (in this case the Intel processor) more secure and safe without the need for changing the rest of the system.

C. Queueing Analysis

The data used in OSATE analysis were subsequently used to construct a queuing model in a simplistic discrete queuing simulator [12]. The exponential distribution was used for the process distribution of the server. The maximum wait time was set to the same value as the average time between arrivals.

TABLE II. SIMULATION RESULTS

	Original Model		**Security FPGA**		**Security Intel**	
	Mean	***Std dev***	***Mean***	***Std dev***	***Mean***	***Std dev***
% utilization	31.86%	0.199	33.46%	0.316	38.19%	0.813
# messages	20191	16.5	19788	375.2	18705	152.7
% lost	0.54%	0.001	2.49%	0.001	8.56%	0.009

An eight hour simulation was run five times and gave the results shown in Table II - comparing the original model with no security component against two options of a model with security implemented on the FPGA or the Intel processor. Adding security component adds processing time to both threads. The results imply that the secure system is losing more messages unable to cope with the amount of traffic. The simulator predicts that there will be a 5% increase in the amount of effort needed to add security enhancements to the message processing functionality in the CACC system on the FPGA processor. There will be an increase of nearly 20% in the amount of effort when the security component is executed on the Intel processor.

D. Results

For the security component bound to the FPGA processor the increase in processor utilization, as computed by AADL

and queuing approach, are 7.8% and 5% accordingly. The respective increase of Intel processor utilization, with security component assigned to Intel, were 13.4% and 19.8%.

The queuing model constructed for the security component bound to the Intel processor was a greater abstraction since only one processor (server) was modeled. We would expect the results to be less accurate than the AADL model in this case. The difference in results between the queuing simulator and the AADL model is partly due to the fact that AADL measures processor utilization in integral numbers. The smallest increment OSATE can measure is 1%, which in this case equates to a 1.9% increase in effort. This may suggest that OSATE is not the best tool to use to measure the increase in effort when more accurate results are required.

Computation of statistical significance (95% confidence interval) between the original model and the models with security enhancements, for both FPGA and Intel processor resulted in the interval does not containing zero. We could thus conclude that the performance of the security enhanced system was statistically different from the original system without security functionality.

V. CONCLUSIONS

Incorporating security into embedded systems involves making trade-offs between the level of security achieved and the resulting decrease in performance. This decision is more critical in embedded system with time, space and power constraints.

Two methods were used to estimate the additional effort required for adding the security functionality in an embedded system. One used the architectural model of the system, while the other obtained results using queuing approach. Such methods, if used early on in the development life cycle, can provide an additional feedback for the system design. Since the results obtained from each method are comparable, our confidence in the obtained results has increased. However, it should be noted that the numerical results have been based on several assumptions due to an exploratory nature of the case study.

A queuing simulator has the added advantage in that it can model stochastic message arrivals, which AADL cannot do. AADL models the functions of the internal system architecture with limited consideration for the interface between the system and external devices. Queuing theory models the system in terms of the external interfaces in terms of the response times and maximum workload. It is interesting to note that the queuing simulator showed that the system as modeled cannot adequately handle the messages in the periods that were assigned. Some, albeit minor, message loss was reported for the speed sensor, brake and gap sensor interfaces. These are the interfaces with the highest periods. This would suggest that the periods specified are too high or that the message processing functionality should be assigned to more than one processor. This is useful information to obtain at the architecture stage of the development process.

The entire CACC system could in theory be modeled as a network of queues since all processing is confined to two components. This would allow more formal analysis to be done in designing the system to handle the estimated amount of message arrivals.

The ability to model the system and analyze it before it is implemented aids in making more intelligent solutions at an early stage of the system development. The validity of the results obtained depends on the accuracy and depth of the information provided to the analysis. If a sufficient level of information is not available at the architectural stage, it is possible to develop a simplistic simulation model in order to obtain estimates of the required data. Message arrivals and servicing could be modeled using queuing theory. Simulators exist which help determine values such as the server utilization and the percentage of requests not serviced. Such approach will also assist to estimate the maximum security that may need to be added to the system to meet the desired performance requirements.

REFERENCES

[1] S.Zhang, N.Zhou, J.Wu, “The Fuzzy Integrated Evaluation of Embedded System Security”, 2008 International Conference on Embedded Software and System Security, IEEE, 2008, pp 157-162

[2] I. Sommerville, "An Integrated Approach to Dependability Requirements Engineering," Proceedings of the 11th Safety-Critical Systems Symposium, Bristol, 2003, pp.3-15

[3] M.Eby, J.Werner, G.Karsai, A.Ledeczi, “Integrating Security Modeling in Embedded System Design”, Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, IEEE, 2007, pp 221-228

[4] P.Feiler, D.Gluch, “The Architecture Analysis and Design Language (AADL): An Introduction”, Technical Note CMU/SEI-2006-TN-011, 2006, Software Engineering Institute.

[5] J.F.Hansson, P.Feiler, J.Morley, “Building Secure Systems Using Model-Based Engineering and Architectural Models”, Crosstalk, September 2008, pp 10-15

[6] K.M.Khan, J.Han, “Composing Security-Aware Software”, IEEE Software, January 2002, pp 34-41

[7] E.Chickowski, OWASP Top 10 - 2010: The Ten Most Critical Web Application Security Risks, (release candidate). OWASP, Retrieved Jan 7, 2011, http://www.channelinsider.com/c/a/Security/Top-10-Most-Critical-Web-App-Security-Risks-298234/

[8] Michigan State University, October 11, 2006, Retrieved March 3,2010, http://www.cse.msu.edu/~chengb/RE-491/Projects/cacc_msu-ford.pdf

[9] A.Dobre, A.King, B.Langpap, W.Stevenson, “Cooperative Adaptive Cruise Control (CACC): Team Project – Interim Report”, SE610: Software Architecture and Design, ERAU, Fall 2009

[10] OSATE: Front-end Processing of AADL Models, Retrieved Apr 15, 2011, http://www.aadl.info/aadl/currentsite/tool/osate.html

[11] J.Deepakumara, H.M.Heys, R.Venkatesan, “Performance Comparison of Message Authentication Code (MAC) Algorithms for the Internet Protocol Security (IPSEC)”, Retrieved March 18, 2010, http://www.engr.mun.ca/~howard/PAPERS/necec_2003b.pdf

[12] G.Darby, “Discrete Event Simulator”, Delphi, Original October 28, 2002, Modified May 18, 2009, Retrieved April 7, 2010, http://www.delphiforfun.org/Programs/discrete_sim.htm