# Integrating Security Modeling into Embedded System Design

Matthew Eby, Jan Werner, Gabor Karsai, Akos Ledeczi
*Institute for Software Integrated Systems*
*Vanderbilt University, Nashville, TN 37235*
*{firstname.lastname}@vanderbilt.edu*

## Abstract

*There is an ever increasing concern about security threats as embedded systems are moving towards networked applications. Model based approaches have proven to be effective techniques for embedded systems design. However, existing modeling tools were not designed to meet the current and future security challenges of networked embedded systems. In this paper, we propose a framework to incorporate security modeling into embedded system design. We've developed a security analysis tool that can easily integrate with existing tool chains to create co-design environments that addresses security, functionality and system architecture aspects of embedded systems concurrently.*

## 1. Introduction

Embedded systems play a crucial role in critical infrastructure in [1], which is essential to national security, success and economic health [2]. There is increasing concern of the security threats on these kinds of embedded systems [3],[4]. Successful attacks have been reported on the US Power Grid [5] and the sewer system of Australia's Maroochy Shire Council [6]. Other incidents such as a worm infection [7] have affected the Davis-Besse Nuclear Power Plant and CSX Railroad Corp. [6]. To address such security threats we need to rethink the embedded software design process.

Model Integrated Computing (MIC) [8] is gaining wide recognition in the field of embedded software design. Models represent embedded software, its deployment platform and its interactions with the physical environment. Models facilitate formal analysis, verification, validation and generation of embedded systems [9]. Hence, this approach is superior to traditional manual software development process. Although, there is modeling tool support for analysis of functionality, performance, power consumption, safety, etc., currently available tools incorporate little if any support for security modeling. As a result, security is looked at only once the complete system has been built. At best, this approach of addressing security in the last stages of development is inefficient taking large amounts of effort to achieve only modest improvements in security. Engineers designing embedded systems usually do not have the experience to address security issues and in many cases are not even aware of the issues [10]. Fixing security vulnerabilities involves releasing patches, which can introduce new problems such as viruses [11] or security vulnerabilities [12]. Still, systems designed without security in mind are intrinsically insecure. Patches can fix specific security vulnerabilities, but do not address poor system architecture.

Many times vulnerabilities are only discovered once they have been exploited. To address these unknown threats, systems can be isolated in private corporate networks using firewalls and intrusion detection systems. But such perimeter defenses, even if they are flawless, cannot protect against insider attacks [13]. In light of this situation, we advocate modeling environments that incorporate security into the early design phase of embedded systems.

One of the few modeling languages with security extensions is UML. Currently available extensions to UML provide: access control [14] [15], fair exchange, assumptions about secrecy, integrity, etc. [16]. There are existing tools that can guarantee some security properties using automated proof verifiers [16]. However, UML based Model Driven Security is not sufficient for design and analysis of embedded systems. We strongly believe that embedded system design can benefit from Domain-Specific Modeling Languages (DSML) as opposed to the one-size-fits-all approach of UML. For example, there is no concept of hardware in UML which makes it ill suited for embedded systems with their diverse hardware architectures. In many embedded applications system resources are scarce. Added overhead for security can have drastic effects on performance. An ideal embedded software development environment will allow the engineer to analyze security and performance tradeoffs based on the hardware platform the system will run on.

## 2. Background and Motivation

MIC can meet the challenges of designing secure embedded systems. A key advantage of the model based approach is the abstraction of the application domain. This abstraction is facilitated through the use of DSMLs. A DSML provides a system designer a set of concepts that are specifically tailored for a certain application domain. In our case, the domain is networked embedded real-time systems, such as process control systems, automotive, avionics and robotics systems. A DSML with the proper level of abstraction hides the inconsequential details of a system while allowing the engineer to shift focus to more important aspects. There are many examples of DSMLs developed for embedded system design in different domains [MILAN [17], SMOLES [18], AADL [17]]. We propose an extension mechanism for DSMLs that adds security concepts similar to UML extensions [14]. By extending embedded system DSMLs, we can add tool support for security analysis, validation, verification and generation. These security tools will extend the large tool chains that already exist for embedded system design.

## 3. General Approach

We will demonstrate a process for integrating security analysis into existing tool chains to create a security co-design environment. The approach taken is to create a common DSML that is used to capture and analyze security properties of systems. The advantage of this approach is that the effort needed develop the security analysis tool is only spent once. Then this tool can be incorporated into existing embedded systems languages with minimal effort. By defining mappings from an embedded system DSML onto the security analysis DSML, we can analyze the security properties the embedded system. Figure 1 illustrates the process of defining mappings from one or more DSMLs onto a language supporting security analysis and feeding the analysis results back to the DSML.

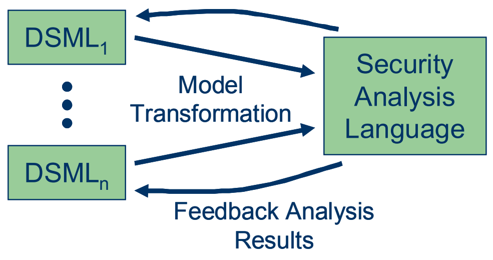

**Figure 1. Mappings from DSMLs to SAL enable security analysis of the DSMLs**

The co-design environment is implemented in the Generic Modeling Environment (GME) [9]. GME is a metaprogrammable tool which facilitates the graphical implementation of DSMLs through the use of metamodels. In this environment, we create a Security Analysis Language (SAL) that enables a user to model and analyze security related properties of embedded systems. (Note that while SAL is technically a DSML, from this point out we use the term DSML only in reference to a language for embedded systems design which we wish to add security analysis capabilities to.) The purpose of this analysis tool is to identify points in the system model that violate certain security requirements and provide useful feedback to the modeler. SAL allows such violations to be identified and remedied at design time before they can be exploited. Currently, SAL supports two types of analyses: information flow analysis and threat model analysis, which are detailed in the following sections.

### 3.1. Information Flow Analysis

The two traditional models for dealing with information flow in systems are the Bell-LaPadula model [19] and the Biba model [20]. Both of these models enforce an access control scheme that defines the rights of a subject to access information. Subjects and information are assigned a security level and a compartment which define what information a given subject is permitted to access. The set of all security levels is an ordered set that can be evaluated as an inequality (i.e. Top Secret > Secret). Compartments are a set that can be evaluated as an inequation (i.e. FBI $\neq$ CIA).

The Bell-LaPadula model deals with confidentiality or secrecy of information in systems. The two properties that the Bell-LaPadula enforces are the Simple Security Property and the * (star) Property. The Simple Security Property states that a subject may not read information that is classified at a level greater than that subject's classification. The * Property states that a subject may not write information to a level less than that subject's classification.

The Biba model deals with integrity of information in systems. Biba also enforces a Simple Security Property and a * Property. The Biba version of the Simple Security Property states that a subject may not read information that is classified at a level less than that subject's classification and the Biba version of the * Property states that a subject may not write information to a level greater than that subject's classification.

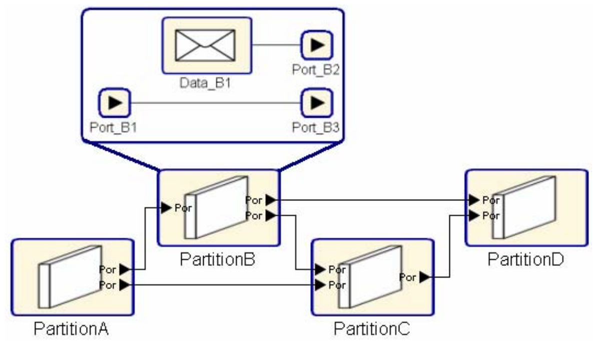

**Figure 2. Partitions and dataflows in SAL**

To analyze the Bell-LaPadula and Biba models, SAL views a system as a set of partitions, a set of data objects contained in each partition and the dataflows inside and across the partitions. Dataflows are represented as connections between input and output ports on a partition. In SAL, partitions are the subjects and are assigned a *security level* and *compartment* attributes. A data object inherits the *security level* and *compartment* classification of its containing partition. SAL allows the *security level* to be an integer value and the *compartment* to be a string value. Our analysis tool treats each data object as the root node in a tree search algorithm. The tool will traverse the dataflow paths originating from a data object and verify that each partition through which that data object flows has a *security level* and *compartment* that permit that partition to access the data object. Bell-LaPadula does not allow information to flow to a lower *security level* while Biba does not allow information to flow to a higher *security level*. When composed, these two models only allow information to flow between partitions with the same *security level*. Applying both models is too restrictive in a system where the designer does not need to restrict access to all data objects. There may be some data objects that have a secrecy requirement but no integrity requirement and vice versa. To provide a less restrictive model, data objects in SAL are assigned two Boolean attributes, *secrecy* and *integrity*. The flow of every data object is evaluated based on the settings of these attributes. When *secrecy* is true the Bell-LaPadula model is enforced and when *integrity* is true the Biba model is enforced on the flow of that data object between partitions. Figure 2 shows a small example model in SAL.

## 3.2. Threat Model Analysis

In a distributed system, partitions may reside on multiple nodes and data is transferred between these nodes over some communication channel. The information flow analysis addresses the movement of data explicitly defined in the system model but does not address covert channels. One such covert channel could be a man-in-the-middle attack on the communication channel. To prevent such attack, the communication channel can be encrypted. Adversary modeling in SAL enables the analysis tool to identify vulnerable channels and determine which encryption algorithms can be used to protect data being transmitted on that channel. Figure 3 illustrates the adversary model.

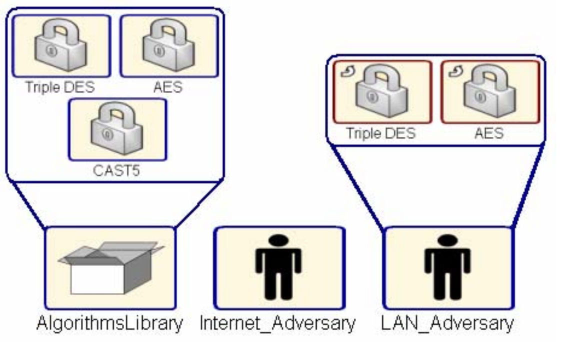

**Figure 3. Encryption algorithms library and adversary models in SAL**

In each system there is a library of encryption algorithms that contains the set of all encryption algorithms that can be used to encrypt a channel. Each system also contains a set of adversary models that define which encryption algorithms are vulnerable in the context of that adversary. Each adversary contains a set of references to algorithms that are defined in the algorithms library. Each reference has an attribute, *maxkeysize*, which means that the referenced algorithm is vulnerable to that adversary if the strength of its encryption is not greater than *maxkeysize*. Together, the encryption algorithm library and adversary models allow our analysis tool to determine which algorithms are safe to use to encrypt information flows. Each information flow in SAL has an attribute, *adversary*, which identifies the adversary model associated with that information flow. Each information flow in SAL also has an *EncryptionAlgorithm* and *KeySize* attribute. For each information flow in the system, the analysis tool checks the *EncryptionAlgorithm* and *KeySize* attribute against the set of encryption algorithms that are vulnerable for the adversary model specified by *adversary*.

## 3.3. Integrating Security Analysis with Existing Tool Chains

Although, there is modeling tool support for analysis of functionality, performance, power consumption, safety, etc., currently available tools incorporate little if any support for security modeling. As a result, security is only addressed once the complete system has been built. We want to leverage the work behind existing tool chains by incorporating security analysis in the system design process. SAL was created to be a reusable tool that can be integrated with multiple tool chains, thus reducing the effort that would be required to develop custom security analysis for each tool chain.

In MIC, a transformation is, in general, a one way function with a domain being the set of all valid models in the original DSML and a range being the set of all valid models in the destination DSML. By defining a transformation that maps models of an embedded system DSML onto SAL, we can perform information flow analysis and threat model analysis on the embedded systems models. In order to define such a transformation, the original DSML must be able to capture those security properties that are need for SAL to provide a useful analysis. In other words, for information flow analysis, the DSML must be able to model the concepts such as data object, dataflow, partition, security level, compartment, secrecy and integrity requirements and for threat model analysis the DSML must be able to model the concepts such as encryption algorithm library, encryption algorithm, adversary model, vulnerable encryption algorithm, encrypted channel and channel adversary. Typically, the DSML will not have all of the concepts needed to create such a transformation. For example, take a DSML built on the synchronous dataflow model of computation [21]. This DSML would have the concepts such as data objects and dataflow, but none of the other security specific concepts. It would be the responsibility of a tool designer to add the ability to capture these security specific concepts in a DSML. The process of extending a DSML to capture security related properties is not as difficult as it might seem. One of the powerful concepts of the MIC approach is easy composition of metamodels to form new languages. By composing the metamodel of a DSML with concepts from SAL, it is relatively easy to form these security specific extensions to an existing language. The tool designer can then create the transformation rules that map models in the DSML onto models in SAL. GME is integrated with the Graph Rewriting and Transformation language (GReAT) [22]. GReAT is built on top of an execution engine (GReAT-E) which can translate models based on transformation rules specified by GReAT. This mapping specification needs to be created only once for a given DSML and then any valid models for that DSML can be automatically transformed into a corresponding SAL model.

Since SAL is only capable of capturing those concepts which are relevant to security analysis, it is not possible to define a transformation from SAL back onto the original DSML. Those concepts which are unique to the original domain are lost in the translation from the DSML to SAL. In order for SAL to provide useful feedback to the user, we introduce the *path* and *id* attributes which belong to partition, information flow, data object, ports, adversary model and encryption algorithm in SAL. *Path* and *id* store the path and the unique ID of an object in the original DSML. When a SAL model is analyzed and security violations are identified, the results will be fed back to the user of the original DSML in the form of an error messages along with hyperlinks that identify at which point in the original model there is a security violation. Using this approach, a user of a DSML will never have to view the SAL model. The transformation to a SAL model, the security analysis and the result feedback form an automated process.

When adding these security specific concepts to a DSML, it is important to consider what they mean in the context of the entire tool chain. Often the design flow includes other tools for such things as functionality, schedulability, power consumption and safety analysis. The division between these different types of analyses is not always clear-cut. Many times decisions made based on one type of analysis can have an impact on the outcome of other types of analysis. One such example in the context of SAL is the use of encryption algorithms. The decision to encrypt a communication channel could have a major effect on the schedulability of the system. Also, if there is a code generator for the DSML, it must be modified to support these security properties (i.e. linking to encryption libraries, enforcing the partition model, etc.). The tool developer who is integrating SAL capabilities to a DSML must address concerns such as making these other tools aware of the impact the security properties will have on the system. Figure 4 shows a typical design flow for performing security analysis with an embedded system DSML.

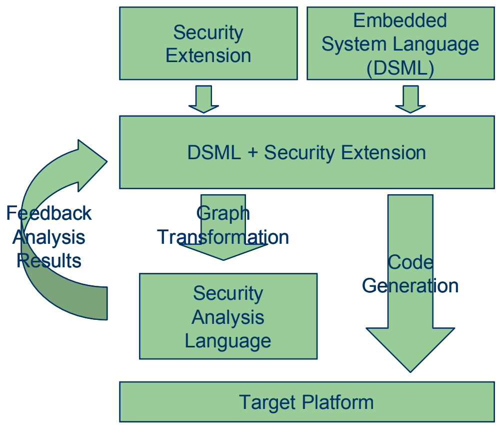
Security Extension
Embedded System Language (DSML)
DSML + Security Extension
Feedback Analysis Results
Graph Transformation
Code Generation
Security Analysis Language
Target Platform

**Figure 4. Typical embedded system design flow with SAL**

## 4. Example: Integrating Security Analysis to an Existing Embedded System Language

As a proof of concept, we have integrated SAL with an existing tool for the design of embedded systems called SMOLES [18]. SMOLES alone does not address security concerns. We show how we add the capability for capturing security specific properties to SMOLES. Models in SMOLES are enriched with these concepts and we create a mapping from SMOLES concepts to objects in SAL. This mapping allows us to create a transformation from SMOLES models to SAL models. We show how results from the analysis of a SAL model can feed back warnings of security violations so that the user can correct the SMOLES model accordingly.

### 4.1. Description of SMOLES

The Simple Modeling Language for Embedded Systems (SMOLES) was designed as a simple modeling language that allows constructing small, embedded systems from components [18]. The components are assumed to be concurrently executing objects that communicate and synchronize with each other. Furthermore, objects can perform blocking I/O operations in which they wait for the result, while other objects can execute. Communication between components means passing data from a source component to a destination component, which is then enabled to run, in order to process the data. In addition to data triggering, periodic timers can also trigger the components. The language consists of components and assemblies. Components are the elementary building blocks, and contain input and output ports, which are used to receive/send data tokens from/to other components. Assemblies contain components, and describe how they are interconnected. Like components, assemblies can have their own input and output ports, and assemblies can contain other assemblies. Assemblies are organized into a hierarchy. The various components and assemblies in the hierarchy communicate with each other through the dataflows, as specified by the designer. SMOLES has a code generation utility that will interpret the model and output C++ code that will execute on top of a small custom dataflow kernel. Figure 5 shows a small example model in the SMOLES DSML.

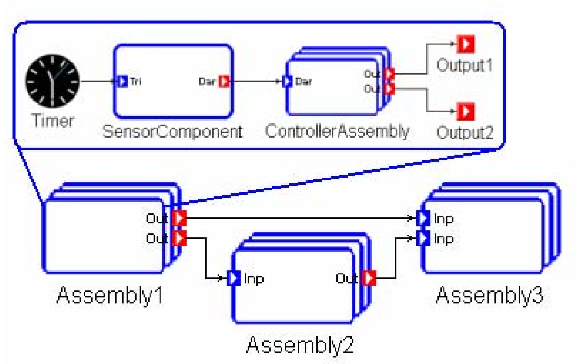

**Figure 5. Example SMOLES model**

### 4.2. Integrating SAL with SMOLES

Since SMOLES does not capture any security properties, we must add the appropriate concepts, so that we can define a transformation from SMOLES to SAL. We call this extended language SMOLES_SEC. SMOLES_SEC allows the modeler to capture the security properties required to perform the two types of analysis that SAL supports, the information flow analysis and threat model analysis.

First, we address those concepts necessary to perform the information flow analysis. SMOLES already has the concept of dataflows but none of the other concepts used in SAL. Assemblies in SMOLES are close to the concept of a partition in SAL. One possible approach would be to add the *security level* and *compartment* attributes to assemblies. However, assemblies are organized in a hierarchy whereas partitions in SAL are not. So, we introduce the idea of partition to SMOLES_SEC. Partitions will have input and output ports which can be connected by dataflow connections. Like in SAL, partitions will have *security level* and *compartment* attributes that define their access rights in the context of the Bell-LaPadula and Biba models. Assemblies will be contained by partitions and will inherit the *security level* and *compartment* of the containing partition. SMOLES has the concept of dataflow; however there is no first class object that is a data object. We assign the *secrecy* and *integrity* attributes to an assembly in SMOLES_SEC and evaluate these attributes against the dataflows originating from that assembly.

Next, we address the concepts necessary to perform the threat model analysis. SMOLES has no concept of encryption algorithms or adversary modeling. We add these concepts to SMOLES_SEC and define them in the same way that they are defined for SAL. In each system, there is an encryption algorithms library with a set of encryptions algorithms. Each system contains a set of adversaries and each adversary contains a set of references to encryptions algorithms. Each encryption algorithm has an attribute, *maxkeysize*. We do not associate an encryption algorithm and adversary model directly with dataflows as it is done in SAL. Rather, SMOLES_SEC can model a deployment diagram where nodes, which represent the execution platform, are connected to other nodes through a link (or bus). A node can be viewed as a set that contain partitions that execute on that node. Likewise, a link can be viewed as a set that contains the dataflows that are transmitted over that link. Each link in SMOLES_SEC has an attribute, *adversary*, which identifies the adversary model associated with that link. Each link in SMOLES_SEC also has an *EncryptionAlgorithm* and *KeySize* attribute. Dataflows inherit the *adversary*, *KeySize* and *EncryptionAlgorithm* of the link that they are transmitted across.

The effect that these extensions for SMOLES_SEC have on tools that were written for the SMOLES languages must be examined. SMOLES has a utility that will generate C++ code from models. This utility is unaware of encryption algorithms and their meaning in the context of SMOLES_SEC. There are two solutions to this problem. Either we define a transformation that will map SMOLES_SEC models back onto SMOLES models or we port this code generator to work with the SMOLES_SEC environment. In our case we choose to port the code generator to the SMOLE_SEC environment. To do this we will need to make some slight modifications such as enabling assemblies to be contained in partitions and linking to a library of encryption algorithms.

### 4.3. Model Transformation from SMOLES_SEC to SAL

Now that the appropriate concepts have been added to extend SMOLE_SEC we are able to define a model transformation that maps SMOLES_SEC models to corresponding models in SAL. Once we have defined these rules, the process of converting SMOLES_SEC models to SAL models will be automated. We have written a small script that can be invoked from the SMOLES_SEC environment. This allows the user of the SMOLES_SEC environment to transform their model into a SAL model in one step, run the information flow and threat model analysis on the SAL model and receive the analysis results.

### 4.4. Example Application in SMOLES_SEC

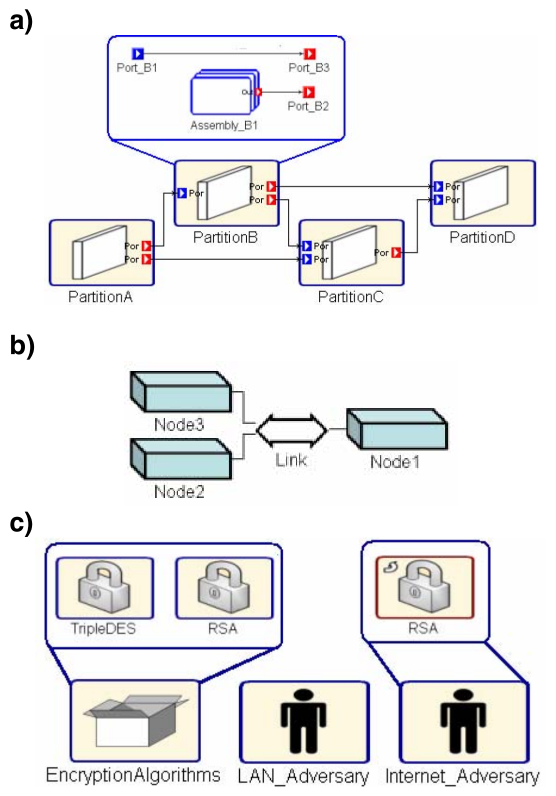

**Figure. 6 SMOLES_SEC example application a) partitions and dataflows, b) deployment diagram, c) threat model.**

An example SMOLES_SEC model is shown in Figure 6. This is a generic application that demonstrates the capabilities of the security analysis. Real applications of this tool will be too large to cover in the scope of this paper. There are four partitions in the system. Partitions A, B, and D have a *security level* of 1 and PartitionC has a *security level* of 2. This means that data objects with a *secrecy* requirement may not flow from PartitionC and those data object with an *integrity* requirement may not flow to PartitionC. In this example, we do not consider partitions with different *compartment* classifications. PartitionB contains an Assembly_B1 that has an *integrity* requirement but no *secrecy* requirement.

Since this assembly is in PartitionB it inherits the security level of 1. Figure 6b shows the deployment diagram. Nodes 1, 2, and 3 are connected by a common link. PartitionA and PartitionB execute on Node1. PartitionC executes on Node2 and PartitionD executes on Node3. All dataflows transfer data across the Link, except the dataflow connecting PartitionA and PartitionB which reside on the same node. Figure 6c shows the threat model of this system. The adversary model of Link is the Internet_Adversary.

First, we will invoke the information flow analysis on this model. Figure 7 shows the error message that we receive for the flow analysis. The assembly in PartitionB has an *integrity* requirement so the dataflows originating from this assembly are evaluated with the Biba model. There is a dataflow that connects PartitionB to PartitionC which represents data objects moving from a low security level to a high security level. This dataflow violates the Biba model. There are a several possible solutions to this error. It is up to the system modeler to determine which solution is appropriate in the context of their system. For this example, we determine that the PartitionB can be classified at a *security level* of 2. When this change is made to the model the security analysis tool does not return any errors.

```
Integrity Requirement Violated --
/SimpleSystem/PartitionB/Assembly_B1 has
an integrity requirement which is
violated by the information flow
connecting
/SimpleSystem/PartitionB/Port_B2 to
/SimpleSystem/PartitionC/Port_C1.
```

**Figure 7 Error message - information flow analysis**

Next, we invoke the threat model analysis. Figure 8a shows the error message we receive. There is an adversary associated with the link so the channel must be encrypted. The error message warns that Link must be encrypted so we set the encryption attributes on Link to 256 bit RSA. Internet_Adversary is capable of breaking RSA with a key size of 256 bits or less. The error message in Figure 8b shows the error message we receive. To fix this error message we increase the key size used to encrypt Link to 512 bits. This fixes the security violation the threat model analysis no longer returns any error messages.

```
a)  Connection not encrypted -- The
    connection at path: /SimpleSystem/Link is
    vulnerable. Please specify an encryption
    algorithm.

b)  Vulnerable Encryption Algorithm -- The
    connection at path: /SimpleSystem/Link is
    vulnerable. The adversary knows RSA up to
    a key size of 256 bits. Please specify a
    larger key size or change the algorithm.
```

**Figure 8 Error messages – threat model analysis**

## 5. Future Work

SAL currently supports modeling of access control policies in the context of the Bell-LaPadula and Biba models. These access control policies are not sufficient for the needs of all applications. We would like for SAL to have the expressiveness to model other types of access control schemes. Another area that needs to be addressed is how to more tightly integrate the security analysis with the other analysis tools available for a DSML. This would allow the designer to look at tradeoffs made based on security properties such as analyzing the tradeoffs between security and performance. We have shown how SAL can be integrated with SMOLES which is a dataflow based language. This leads to a simple mapping from SMOLES to SAL. There needs to be work done to look at how security analysis can be integrated with other classes of DSML such as those based on control flow.

## 6. Conclusion

Model driven security approaches have been successfully used in various industrial, governmental and financial applications. Model-Integrated Computing has proven to be a valuable tool in embedded systems design process. We have demonstrated a security analysis tool that is capable of analyzing the flow of data objects through a system and identifying points in a distributed system that are vulnerable to attack. We have outlined a method for composing this type of security tool with existing tool chains for DSMLs. This approach leverages the development efforts that have gone into design of tool suites for existing embedded system DSMLs. Creating a separate analysis language for security properties allows reuse of this tool for multiple DSMLs. The example application shown is a proof of concept that demonstrates the potential of integrating security modeling capabilities with existing languages.

## 7. Acknowledgement

This work was supported in part by TRUST (The Team for Research in Ubiquitous Secure Technology), which receives support from the National Science Foundation (NSF award number CCF-0424422).

## 8. References

[1] Fernandez, J. D. and Fernandez, A. E. 2005. *SCADA systems: vulnerabilities and remediation*. *J. Comput. Small Coll.* 20, 4 (Apr. 2005), 160-168.

[2] Amin, M. North America's electricity infrastructure: are we ready for more perfect storms? Security & Privacy Magazine, IEEE Volume 1, Issue 5, Sept.-Oct. 2003 Page(s):19 - 25

[3] Kocher, P., Lee, R., McGraw, G., and Raghunathan, A. 2004. Security as a new dimension in embedded system design. In *Proceedings of the 41st Annual Conference on Design Automation* (San Diego, CA, USA, June 07 - 11, 2004). DAC '04. ACM Press, New York, NY, 753-760.

[4] Ravi, S., Raghunathan, A., Kocher, P., and Hattangady, S. 2004. Security in embedded systems: Design challenges. *Trans. on Embedded Computing Sys.* 3, 3 (Aug. 2004), 461-491.

[5] F. Schneider, editor. *Trust in* .National Academy Press, Washington, DC, 1999. Available at http://www.nap.edu/readingroom/books/trust/

[6] Andrew Hildick-Smith, *Security for Critical Infrastructure SCADA systems.* August 24 2005. SANS Institute. Available at http://www.sans.org/rr/whitepapers/warfare/1644.php

[7] Weaver, N., Paxson, V., Staniford, S., and Cunningham, R. 2003. *A taxonomy of computer worms*. In *Proceedings of the 2003 ACM Workshop on Rapid Malcode* (Washington, DC, USA, October 27 - 27, 2003). WORM '03. ACM Press, New York, NY, 11-18

[8] Sztipanovits, J.; Karsai, G. *Model-integrated computing,* Computer Volume 30, Issue 4, April 1997 Page(s):110 – 111

[9] Karsai, G., Sztipanovits, J., Ledeczi, A., Bapty, T.: *"Model-Integrated Development of Embedded Software,"* Proceedings of the IEEE, Vol. 91, No.1., pp. 145-164, January, 2003

[10] Tsipenyuk, K.; Chess, B.; McGraw, G.; *Seven pernicious kingdoms: a taxonomy of software security errors* Security & Privacy Magazine, IEEE Volume 3, Issue 6, Nov.-Dec. 2005 Page(s):81 - 84

[11] Microsoft TechNet. Information About Virus-Infected Hotfixes. April 25 2001. Available at http://www.microsoft.com/technet/ security/alerts/info/vihotfix.mspx

[12] Microsoft TechNet. Microsoft Security Bulletin MS06-007. February 14 2006. Available at http://www.microsoft.com/technet /security/Bulletin/MS06-007.mspx

[13] Kevin Poulsen, *Slammer worm crashed Ohio nuke plant network,* August 19 2003. Available at http://www.securityfocus.com/ news/6767

[14] Jürjens, J. 2005. *Sound methods and effective tools for model-based security engineering with UML*. In *Proceedings of the 27th international Conference on Software Engineering* (St. Louis, MO, USA, May 15 - 21, 2005). ICSE '05. ACM Press, New York, NY, 322-331.

[15] Basin, D., Doser, J., and Lodderstedt, T. 2003. *Model driven security for process-oriented systems*. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies* (Como, Italy, June 02 - 03, 2003). SACMAT '03. ACM Press, New York, NY, 100-109.

[16] Jan Jürjens, *Secure Systems Development with UML*, Springer-Verlag, 2004

[17] Available from the Authors

[18] Szemethy, T. and Karsai, G. 2004. Platform modeling and model transformations for analysis. *Journal of Universal Computer Science 10*, 10, 1383–1406.

[19] D.E. Bell and L.J. LaPadula. "Secure Computer Systems: Mathematical Foundations and Model," Mitre Corp. Report No. M74-244, Bedford, Mass., 1975.

[20] K.J. Biba, "Integrity Considerations for Secure Computer Systems," Mitre Corp. Report TR-3153. Bedford. Mass., 1977.

[21] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proc. IEEE*, vol. 75, pp. 1235–1245, Sept. 1987.

[22] Agrawal, A., Karsai, G., Ledeczi, A.: An End-to-End Domain-Driven Software Development Framework. Conference on Object Oriented Programming Systems Languages and Applications, 2003.