Modeling Security Aspects in Distributed Real-Time Component-Based Embedded Systems

Mehrdad Saadatmand
Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University, Västerås, Sweden
mehrdad.saadatmand@mdh.se

Thomas Leveque
Orange Labs
Orange, Meylan, France
thomas.leveque@orange.com

***Abstract*—Model Driven Engineering (MDE) and Component Based Software Development (CBSD) are promising approaches to deal with the increasing complexity of Distributed Real-Time Critical Embedded Systems. On one hand, the functionality complexity of embedded systems is rapidly growing. On the other hand, extra-functional properties (EFP) must be taken into account and resource consumption must be optimized due to limited resources. However, EFP are not independent and impact each other. This paper introduces concepts and mechanisms that allow to model security specifications and derive automatically the corresponding security implementations by transforming the original component model into a secured one taking into account sensitive data flow in the system. The resulted architecture ensures security requirements by construction and is expressed in the original meta model; therefore, it enables using the same timing analysis and synthesis as with the original component model.**

***Index Terms*—Model-Driven Development, Component model, Embedded systems, Security.**

I. Introduction

Design of real-time embedded systems is a challenging task. This is mainly due to the complexity of these systems that originate from different range of extra-functional properties that they need to satisfy, while taking into account their limitations of resources. This gets even more complex when we realize that the extra-functional properties in these systems are tightly inter-connected and cannot be considered in isolation [1]. Due to the nature of real-time embedded systems (e.g. usage of sensors and actuators and interacting with the environment), timing properties in these systems are of utmost importance. However, implications of other properties and aspects, such as security, on timing properties should also be taken into account to ensure a correct design.

Security aspects in embedded systems are gaining more and more attention especially when they are distributed. Introducing security in the design of an embedded system has impacts on other properties such as timing, performance, memory usage, and energy consumption. On the other hand, the usage and hostile operational environment of embedded systems makes them also exposed to specific attacks that might not be that relevant for other systems [2]. For example, smart cards and wireless sensor networks which are physically exposed, are quite tamper-prone compared to a bank database server which is protected from access and isolated physically in a separate room. There is a need to include security properties in the design of a distributed real-time embedded system while still enabling prediction of timing properties.

Model-Driven Engineering (MDE) and Component-Based Development (CBD) are two promising approaches that can be used orthogonally to alleviate the design complexity of real-time embedded systems. Component-Based Development enables reuse of already existing software units (components) by developing a system as an assembly of components instead of building the system from scratch. Model-Driven Engineering, on the other hand, helps to raise the abstraction level and perform analysis at earlier phases of development. This enables the identification of problems in the system design before reaching the implementation phase [1], [3], [4].

Using benefits of these two approaches, we propose to specify security needs as annotations on the ProCom component model [5] and derive the equivalent component model which implements the security specification. Using our approach, the designer specifies sensitive data flows with required security properties and selects an implementation strategy to fulfill these requirements. Based on this information, a component model conforming to the original meta-model is generated which satisfies the security specification and the Worst Case Execution Time (WCET) of the resulted components is computed. Therefore, same tools and analyses such as timing analysis and synthesis are applicable for the original component model and the derived one. As a result, timing implications of specified security properties are predictable. This approach facilitates system designers in bringing security aspects into the design model.

The remainder of the paper is structured as follows. In Section II, motivation of this work and security challenges in the design of embedded systems are discussed. Section III, introduces Automatic Payment System as an example of distributed real-time embedded systems with security requirements. The suggested approach is described in detail in Section IV. Implementation and analysis results are explained in Section V. Section VI discusses related work. Finally , we conclude the paper and describe future work and directions of this work in Section VII.

II. Motivations

Security is an aspect that is often neglected in the design of

embedded systems. However, the use of embedded systems for critical applications such as controlling power plants, vehicular systems control, and medical devices makes security considerations even more important. This is due to the fact that there is now a tighter relationship between safety and security in these systems (refer to [6] for definitions of security and safety and their differences). Also because of the operational environment of embedded systems, they are prone to specific types of security attacks such as physical and side channel attacks [7] that might be less relevant for other types of systems. Increase in use and development of distributed networked and connected embedded devices also opens them up to new types of security issues. Features and devices in a car that communicate with other cars (e.g. the car in front) or traffic data centers to gather traffic information of roads and streets, use of mobile phones beyond just making phone calls and for purposes such as buying credits, paying bills, and transferring files (e.g. pictures, music,etc.) are tangible examples of such usages in a distributed networked environment.

Because of the constraints and resource limitations in embedded systems, satisfying a non-functional requirement such as security requires careful balance and trade-off with other properties and requirements of the systems such as performance and memory usage. Therefore, introducing security brings along its own impacts on other aspects of the systems. This further emphasizes the fact that security cannot be considered as a feature that is added later to the design of a system and needs to be considered from early stages of development and along with other requirements. Considering the characteristics of embedded systems, major impacts of security features in these systems are on performance, power consumption, flexibility and maintainability, and cost [7]. Therefore in the design of embedded systems, implications of introducing security decisions should be taken into account and analyzed.

Today, security is mostly taken into account at code level which requires detailed knowledge about security mechanisms while there is a need to raise the abstraction level to deal with the complexity of security implementation.

III. Automatic Payment System

An example of a distributed embedded system with real-time and security requirements is the Automatic Payment System that is being designed for toll roads as depicted in Figure 1. In this system, the purpose is to reduce the waiting time and thus traffic that is caused by that at tolling stations.

For each tolling station, a camera is used that detects a vehicle as it approaches the station (e.g. at 100/200 meter distance), and scans and reads its license plate information. This information is passed to the payment station subsystem which then sends the toll fee to the vehicle through a standardly defined wireless communication channel. The vehicle shows the amount to pay to the driver through its User Interface (UI) and the driver inserts a credit card and accepts the payment to be done. The credit card number is then sent securely to the payment station which then performs the transaction on

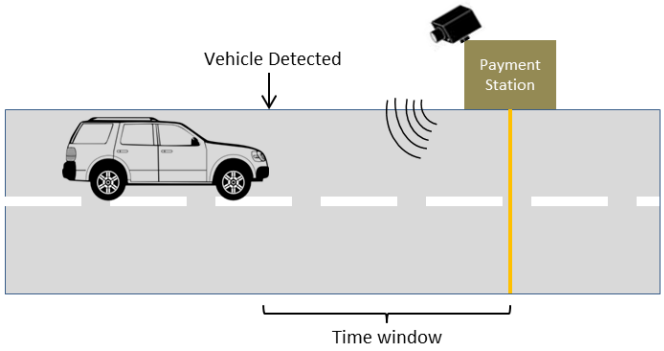

Figure 1. Automatic Payment System for Toll Roads.

it through a (third party) merchant (through a wired Internet connection at the station). The driver is then notified about the success of the transaction and receives an *OK* message to go. Different objects in this system and the interactions between them are shown in Figure 2.

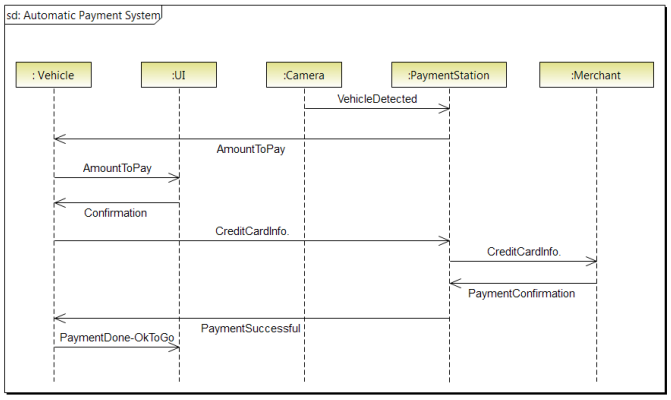

Figure 2. Automatic Payment System for Toll Roads.

The system should perform all these operations in a certain time limit in order to allow a smooth traffic flow. Such time constraints can be calculated considering the specifications of camera and required time for detection, traffic and safety regulations (e.g, allowed speed), and other similar factors. For example, if the vehicle is detected at 100 meter distance from the station, and the allowed speed at that point is 20 km/h, then the system has a strict time window during which it should be able to store the vehicle information, establish communication, and send the payment information to it. Different scenarios can happen in this system. For example, it could happen that the driver/vehicle fails to provide credit card information, or the credit card is expired. In this case, the system can log the vehicle information in a certain database and send the bill later to the owner, or even it can be set to not open the gate for the vehicle to pass and also show a red light for other cars approaching that toll station to stop. Besides the mentioned timing constraints that exist in this system, the communication between different nodes and transfer of data need to be secured

and protected. In this system, we have the following security requirements:

1) Sensitive data such as credit card information should not be available to unauthorized parties.
2) The vehicle only accepts transactions initiated by the payment station.

To achieve these requirements, the station needs to authenticate itself to the vehicle so that the vehicle can trust and send the credit card information. Moreover, sensitive information that is transferred between different parts should also be encrypted.

IV. Approach

A. General Approach

Our approach aims to introduce security concerns in the design of embedded systems. The main idea is to identify the different data entities which need to be confidential and/or that the sender must be authenticated. From the specification of security needs at data level and physical platform level, a transformation is applied on the component model in order to add security implementation. The resulted system ensures the security specification.

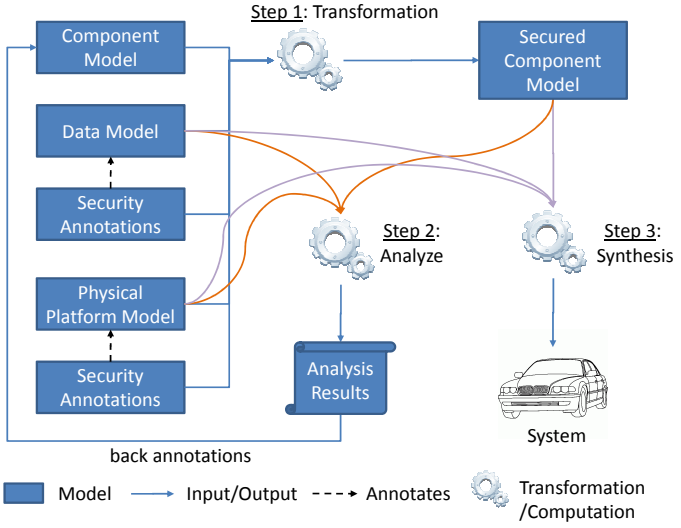

Figure 3. Approach Process.

Figure 3 shows the overall process of the approach. The system is described in several models based on ProCom component model [5]. While the approach is not ProCom specific, it relies on the main assumption that a component model transformation to introduce security implementation exists which has been proved in ProCom but remains as future work for other component models. In the remaining, we will refer to component model to represent the architecture model. Security needs are specified as annotations on top of the data model and the physical platform model. A benefit of the ProCom component model is his ability thanks to its attribute framework to extend any element with additional attribute. We use this mechanism to specify our annotations. Having this information in the model, the following steps are then performed:

1) The component model which specifies the functional and extra-functional part of the system is transformed in a functional equivalent model with added security implementations;
2) Analysis can be performed on the secured component model whose result is back annotated (for example with timing properties) to the original model; and
3) Finally, the system is synthesized.

The considered process is iterative and allows to refine security specification after evaluating resulted system properties such as timing properties.

B. ProCom Component Model

While the approach principles seem to be component model generic, we implemented it using ProCom. The ProCom component model targets distributed embedded real-time system domain. In particular, it enables to deal with resource limitations and requirements on safety and timeliness concerns. ProCom is organized in two distinct layers that differ in terms of architectural style and communication paradigm. For this paper, however, we consider only the upper layer which aims to provide a high-level view of loosely coupled subsystems. This layer defines a system as a set of active, concurrent subsystems that communicate by asynchronous message passing, and are typically distributed. Figure 4 shows ProCom design of the Automatic Payment System example.

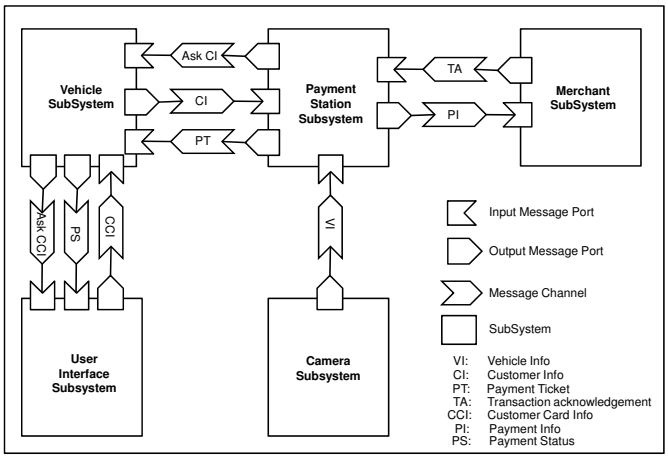

Figure 4. Component Model of the System using ProCom.

A subsystem can internally be realized as a hierarchical composition of other subsystems or built out of entities from the lower layer of ProCom. Figure 5 shows the implementation of the subsystem E as an assembly of two component C1 and C2. Data input- and output ports are denoted by small rectangles, and triangles denote trigger ports. Connections between data- and trigger ports define transfer of data and control, respectively. Fork and Or connectors, depicted as small circles specify control over the synchronization between the subcomponents.

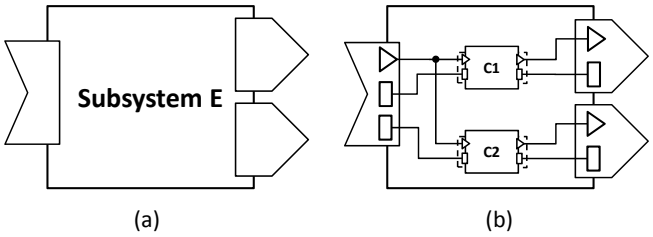

Figure 5. ProCom SubSystem Implementation.

TABLE III
MAPPING BETWEEN DATA ENTITIES AND MESSAGE CHANNELS.

Message Channel	Data Entities
AskCI	AskCI
CI	CustomerInfo
PT	PaymentTicket
AskCCI	AskCCI
PS	PaymentStatus
CCI	CreditCard
VI	VehicleNumber, VehicleType
TA	CCNumber, AmountToPay, PaymentStatus
PI	PaymentRequest

C. Data Model

As components are usually intended to be reused, data should also be reused. To this end, we propose to extend the approach described in [8]. Every data entity is stored in a shared repository. Its description contains its type (String, int...), its maximum size and its unit. A data entity can also be a composite entity defined as a list of data entities. Table I and Table II show data entities of our example. As described in the last section, subsystems communicate through asynchronous message passing represented by message channels. A message channel is associated with a list of data entities which defines the message content. Table III presents mapping between message channels and data entities for our example. We can observe that the same data entity can be used several times in different message channels. The mapping between data ports of message ports and data entities is based on naming convention which enables to distinguish between data ports that require to encrypt/decrypt their data and those that do not. We call data model the set of data entities which are used in the related design.

TABLE I
PRIMITIVE DATA ENTITIES.

Data Entity	Type	Max Size	Unit
CCNumber	String	16	byte
ExpirationDate	String	4	byte
AskCI	Empty	0	byte
AskCCI	Empty	0	byte
PaymentStatus	boolean	1	byte
VehicleNumber	String	20	byte
VehicleType	Enum	8	byte
AmountToPay	float	4	euro

TABLE II
COMPOSITE DATA ENTITIES.

Data Entity	Contains
CreditCard	CCNumber, ExpirationDate
CustomerInfo	VehicleNumber, CreditCard
PaymentTicket	AmountToPay, PaymentStatus
PaymentRequest	AmountToPay, CreditCard

D. Physical Platform And Deployment Modeling

The physical entities and their connections are described in a separate model called Physical Platform Model (see Figure 6). This model defines the different Electronic Computation Unit (ECU) called Physical Node including their configuration such as processor type and frequency, the connections between physical nodes and the physical platforms which represents a set of ECU fixed together.

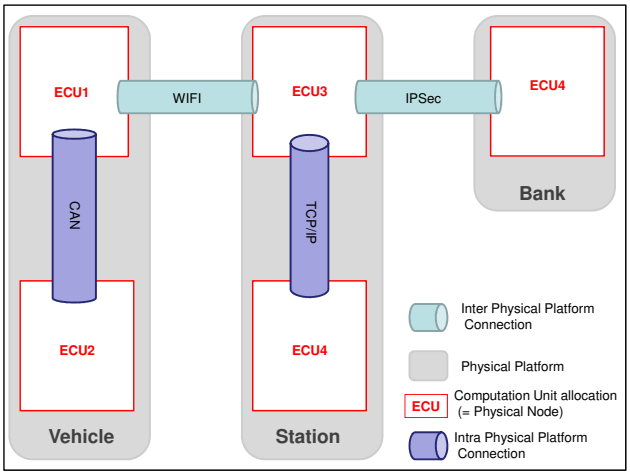

Figure 6. Physical Platform Model of the System.

ProCom system deployment is modeled in two steps, introducing an intermediate level where subsystems are allocated to virtual nodes that, in turn, are allocated to physical nodes. In a similar way, message connections are allocated to virtual message connections which, in turn, are allocated to physical connections. Figure 7 defines the physical platform and related mapping of Automatic Payment System model. To simplify the example, we assume one to one mapping between virtual node and physical node.

E. Security Properties

Instead of defining the security properties on the architecture, i.e. the component model, we propose to annotate the data model and compute the required security properties on the architecture, based on these security requirements. It is an original part of our approach where designer can think about sensitive data without considering the architecture models. The designer applies security properties to identify and annotate sensitive data in the system, which require to be protected using some security mechanisms (e.g., confidentiality and

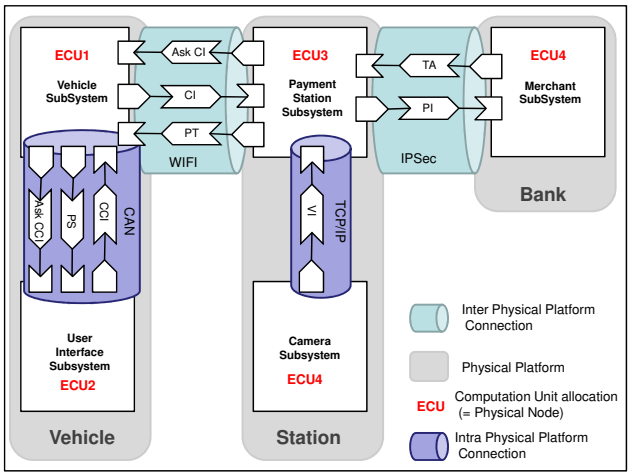

Figure 7. Deployment Model of the System depicting allocation to Physical Platforms.

encryption, authentication, integrity, etc.). We consider two types of security properties:

- **Confidentiality** ensures that the considered information can not be read by any external person of the system; and
- **Authentication** which ensures that the considered information comes from the expected sender.

Table IV shows security annotations associated to data entities for our example. In addition to security properties on

TABLE IV
DATA ENTITY SECURITY PROPERTIES.

Data Entity	Security properties
CCNumber	Confidentiality
VehicleNumber	Authentication
AskCI	Authentication
AskCCI	Authentication
PaymentRequest	Authentication
PaymentStatus	Authentication

the data model, we define the security properties related to the physical platform which are independent of any application:

- **Exposed** defines that the physical platform is potentially accessible to external persons and that they may be able to open it and modify physical parts.
- **NotAccessible** defines that the physical platform is not considered as accessible to unauthorized persons.

In a similar way, physical connections are annotated:

- **NotSecured** defines that the physical connection protocol does not implement reliable security.
- **Secured** defines that the physical connection is considered as secured due to its intrinsic security implementation.

Using these properties, the responsible of the physical platform annotates physical entities and the physical connections between them in the platform model. Thanks to these annotations, we can deduce which parts do not need additional security implementations if it is already provided. For example, if a link is established using mere TCP/IP, it is annotated as NotSecured, while in case that IPSec protocol suite is used for a link, that link is annotated as Secured. This means that the link is considered trusted and already secured, and no security component is necessary to be added for the link. Table V shows the security properties of Automatic Payment System physical platforms.

TABLE V
SECURITY PROPERTIES OF PHYSICAL ENTITIES.

Physical Platform or Connection	Security properties
Vehicle	Exposed
Station	NotAccessible
Bank	NotAccessible
WIFI	NotSecured
IPSec	Secured
TCP/IP	NotSecured
CAN	NotSecured

F. Cost of Security Implementations

To satisfy the identified security properties in the system, different security mechanisms, namely encryption/decryption algorithms in this paper, can be used. As stated before, using a security mechanism in the system has its own costs in terms of timing and performance, power consumption and so on. Therefore, choosing an appropriate security mechanism is critical in order to ensure the satisfaction of timing requirements of the system while fulfilling the security requirements. For this purpose, and to take into account the timing costs of different security mechanisms, we rely on the results of studies such as [9] that have performed these cost measurements. Based on such methods, we assume the existence of such timing measurements for the platforms used in our system in the form of the Table VI. We assume that execution time can be computed knowing targeting platform, algorithm, key size and data size. We suggest to provide a timing estimation toolkit which enable to provide estimate based on measurements. It can be observed that depending on the targeted platform, some algorithms may not be supported.

TABLE VI
EXECUTION TIMES AND STRENGTH RANKING OF DIFFERENT SECURITY ALGORITHMS FOR A SPECIFIC PLATFORM

Strength Rank	Algorithm	Key Size	ET-P1	ET-P2	ET-Pn
1	AES	128	NS	480	...
2	3DES	56	292	198	...
3	DES	56	835	820	...
...					

(ET-Px: Executime Time on Platform x in bytes per second, NS: Not Supported on corresponding platform)

Table VI shows estimates based on results presented in [9]. These results are related to a specific platform. We do not aim to explain how to get such table. However we assume that it is possible to get such estimates.

G. Security Implementation Strategy

As mentioned previously, based on the selected strategy, a security mechanism is chosen from the table and the components implementing it are added to the component model.

The user can then perform timing analysis on the derived component model to ensure that the overall timing constraints hold and are not violated. We propose several strategies to help choosing between all possible security implementations:

- The **StrongestSecurity** strategy selects the strongest security implementation available on the platforms;
- The **StrongestSecurityAndLimitImplemNb** strategy selects the strongest security implementation available on the platforms which ensures that we use as few as possible different security implementations since each message channel can use a different encryption algorithm;
- The **LowestExecTime** strategy selects the security implementation available on the platforms which has the lowest execution time;
- The **LowestExecTimeAndLimitImplemNb** strategy selects the lowest execution time implementation available on the platforms which ensures that we use as few as possible different security implementations; and
- The **StrongestSecuritySchedulable** strategy selects the strongest security implementation available on the platforms where the system remains schedulable.

The selection is driven by the fact that the same algorithm must be used for the sender and receiver components which may be deployed on different platforms which in turn may not support the same algorithms. The StrongestSecuritySchedulable strategy is hard to implement and will be part of our future works. However, it is the most interesting one. More complex security implementation strategies can be considered but are not covered by this paper. In particular, our future works will try to define required security strength associated to data and message channels.

H. Transformation

The transformation is performed in four steps:

1) First, we identify the part of message which needs to be confidential or authenticated and on which communication channels;
2) Then, we add components in charge of encryption, decryption of the identified communication channels;
3) Then, the strategies are used to choose which encryption algorithm to use and generate the code of added components; and
4) Finally, the Worst Case Execution Time (WCET) of added components is estimated.

It relies on the following assumptions:

- The confidentiality is ensured using asymmetric keys; and
- The authentication is ensured using electronic certificates.

The transformation aims to ensure that data decryption is performed once and only once before that data will be consumed and that data encryption is performed once and only once when a message should be sent. To illustrate the algorithm, let's consider the example in Figure 8. We assume that only data D1 need to be confidential. The pseudo algorithm of the transformation is described in Listing 1.

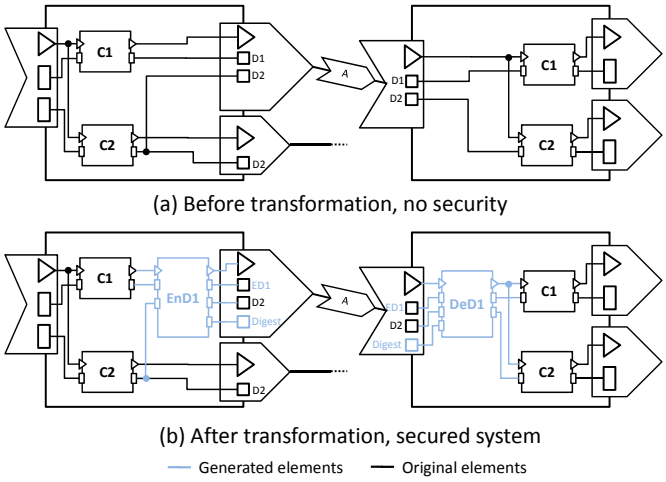

Figure 8. Transformation.

Listing 1. Transformation Pseudo Algorithm

```
msgToSecure = {}
for all channels M in component model {
 P = M.allocatedPhysicalChannel;
 if ((M.getConfidentialData() <> {}) or
     (M.getAutheticatedData() <> {}) and
   (P.isNotSecured()) and
   ((P.isIntraPlatform() and
     P.sourcePort.platform.isExposed()) or
    (P.isInterPlatform()))
       add M in msgToSecure;
}

for all M in msgToSecure {
 P = M.allocatedPhysicalChannel;

 Source = M.sourcePort;
 EnD = create component
       with same ports as Source;
 if (M.getAutheticatedData() <> {})
    add one output port Digest to EnD
    add one input port Digest to Source
 EnD.inConnections = Source.inConnections;
 create connections where EnD.outPorts
    are connected to corresponding
    Source.inPorts;
 generate EnD implementation code

 Dest = M.destPort;
 DeD = create component
       with same ports as Dest;
 if (M.getAutheticatedData() <> {})
    add one output port Digest to Dest
    add one input port Digest to DeD
 DeD.outConnections = Dest.outConnections;
 create connections where Dest.outPorts
    are connected to corresponding
```

```
    DeD.inPorts;
  generate DeD implementation code
}
```

Encryption/Decryption (in EnD1 and DeD1) is done only for confidential data while other data are just copied. An additional port is used to send digest used for authentication. The decryption component (DeD1) ensures that all message data will be available at the same time thought output data ports. This implementation ensures the original operational semantic of the component model. Then, the security strategy is used to choose which encryption/decryption algorithm must be used and what its configuration will be.

V. Implementation

This approach has been experimented partially in PRIDE, the ProCom development environment. The feasability at model level of the approach has been validated while the code generation part remains as future works. The security annotations have been added using the Attribute framework [10] which allows to introduce additional attribute to any model element in ProCom. The model transformation has been implemented using a QVTo [11] transformation plugged at the end of the process described in [12]. These experiments aims to show the benefits at design level of the approach where timing properties of the overall system can be analysed. The current implementation only supports the LowestExecTime and StrongestSecurity strategies. The analysis of estimate accuracy is out of the scope of this paper.

This paragraph presents some ideas to generate code of security components. In order to keep the approach generic, we intend to let certificate specification and other encryption algorithm specific parameters to be filled in the generated code. One generator is associated for each algorithm. The suitability for timing analysis of the generated component code need to be planned but at least will allow for measurement based timing analysis as any other ProCom component. While the system functionality remains the same, the system needs to react to authentication errors. This problem could be partially solved by letting opportunity to the developer to add code to manage authentication errors in the generated code which leads to define what must be the output data in this specific case.

VI. Related Work

With the growing complexity of real-time embedded systems, management of run-time data in these systems has become more important than before and has gained more attention. It has become extremely hard for one person to keep track of all data that is passing through different parts of the systems. Currently, most design methods based on component models focus on functional structuring of the system without considering data flow meaning and semantics [8]. [8] introduces a data-centric approach which models data and proposes to use real time database for data management at runtime in real-time embedded systems. It introduces an architectural view for data entities to complement component-based views. Unfortunately, it does not address extra-functional properties such as security. Our work follows a similar approach to model data entities as a basis to define security specification.

For modeling security features in general, several solutions have been offered such as UMLsec [13] that is a UML profile for the specification of security relevant information in UML diagrams. It is one of the major works in this area and also comes with a tool suite which enables evaluation of security aspects and their violations. There are other similar approaches that have narrower focus like SecureUML [14] that enables modeling of role-based access controls. However, modeling security requirements in isolation (from other aspects of the system) is not enough and become problematic to predict impact on other extra-functional properties especially for real-time embedded. There are works such as [2] and [7] that discuss security issues unique to embedded systems. In [15] we have proposed and discussed benefits of extending MARTE [16] modeling language with security annotations to cover the modeling needs of embedded systems. This work focused on providing UML stereotypes to specify confidentiality property of message communication and related timing estimates. Contrasting, our work models confidentiality and authentication properties at higher abstraction level enabling the designer to focus on sensitive data without thinking about the security implementation which will be automatically generated.

In [17], a method is introduced to specify security requirements on UML models and check their satisfaction by relating model-level requirements to code level implementations. UMLsec is used to include security requirements at model level, and JML annotation language is used to relate code blocks back to the security requirements specification, therefore enables evaluation of security requirement assurance. While this work is also an MDE based approach for defining security requirements, it does not provide timing impacts of security implementation and does not automatically derive security implementation.

The work described in [18] considers security issues in model-based development of service-oriented applications. It proposes a process meta-model for modeling service orchestration as a workflow and another one for modeling high-level security properties. In addition, a more detailed model is used to specify low level security specification and the tool generates the security implementation from these models. Our approach is similar to this work in the sense that it aims to define security properties at high level of abstraction. While they focus on automatic service selection and other security properties such as integrity, our objective is to compute timing impact of security implementation and target another application domain. In addition, we identify sensitive part of messages which need to be secured.

The work in [19], highlights the need to identify sensitive data and introduces an extension to include security concerns as a separate model view for web-services based on Web-Services Business Process Execution Language (WS-BPEL). However, it does not take into account consequences of security design decisions on timing.

Studies [9], [20] are two examples of works that measure the costs of security algorithms. [9] provides performance comparisons of several encryption algorithms, and [20] compares energy consumptions of encryption algorithms on two sensor motes used for building wireless sensor networks. While the focus in these works is not on system design, MDE, and CBD methods, in our work they serve as hints and examples on how to get costs of security algorithms.

VII. Conclusion

Modeling of data in embedded systems is mainly done by specifying the type of data, while the semantics of transferred data is needed for security concerns such as identification of sensitive data. Security, as a non-functional requirement, spans different parts of a system and needs to be modeled at all levels (component architecture, data model, physical platform...). In this paper, we presented an approach which allows to define security specification of embedded systems at high level of abstraction and to produce automatically the security implementation.

Our contributions are

- To propose to model data semantic on top of ProCom architecture model;
- To propose a way to model security properties and needs as annotations on the different models;
- To provide a transformation of component model which ensures by construction security specification; and
- To enable timing analysis on secured component model by computing timing estimates of security components.

All these features help system designers to focus more on system architecture and timing properties which are critical in real-time embedded systems, and at the same time, consider and apply security mechanisms in the design models, without the need to have deep knowledge about how to implement different security mechanisms. This also contributes to the aim and trend of bringing security considerations in higher levels of abstraction.

As future works, we aim to provide analysis to compute security system properties such as robustness of the system to a specific attack, to provide an algorithm for the StrongestSecuritySchedulable, evaluation on an industrial use case and to consider other extra functional properties such as power consumption of security implementations and to provide trade-off analysis between security properties and other extra-functional properties.

References

[1] M. Voelter, C. Salzmann, and M. Kircher, "Model driven software development in the context of embedded component infrastructures," in *Component-Based Software Development for Embedded Systems*, ser. Lecture Notes in Computer Science, C. Atkinson, C. Bunse, H.-G. Gross, and C. Peper, Eds., vol. 3778. Springer Berlin / Heidelberg, 2005, pp. 143–163.

[2] S. Gürgens, C. Rudolph, A. Maña, and S. Nadjm-Tehrani, "Security engineering for embedded systems: the secfutur vision," in *Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systems*, ser. S&D4RCES '10. New York, NY, USA: ACM, 2010, pp. 7:1–7:6.

[3] M. Torngren, D. Chen, and I. Crnkovic, "Component-based vs. model-based development: a comparison in the context of vehicular embedded systems," in *Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference on*, aug.-3 sept. 2005, pp. 432 – 440.

[4] B. Selic, "The pragmatics of model-driven development," *IEEE Software*, vol. 20, pp. 19–25, September 2003.

[5] S. Sentilles, A. Vulgarakis, T. Bures, J. Carlson, and I. Crnkovic, "A Component Model for Control-Intensive Distributed Embedded Systems," in *Proceedings of the 11th International Symposium on Component Based Software Engineering (CBSE2008)*, M. R. Chaudron and C. Szyperski, Eds. Springer Berlin, October 2008, pp. 310–317.

[6] E. Albrechtsen, "Security vs safety," NTNU - Norwegian University of Science and Technology http://www.iot.ntnu.no/users/albrecht/, Accessed: May 2011.

[7] P. Kocher, R. Lee, G. McGraw, and A. Raghunathan, "Security as a new dimension in embedded system design," in *Proceedings of the 41st annual Design Automation Conference*, ser. DAC '04. New York, NY, USA: ACM, 2004, pp. 753–760, moderator-Ravi, Srivaths.

[8] A. Hjertström, D. Nyström, and M. Sjödin, "A data-entity approach for component-based real-time embedded systems development," in *Proceedings of the 14th IEEE international conference on Emerging technologies & factory automation*, ser. ETFA'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 170–177.

[9] A. Nadeem and M. Javed, "A performance comparison of data encryption algorithms," in *Information and Communication Technologies, 2005. ICICT 2005. First International Conference on*, aug. 2005, pp. 84 – 89.

[10] S. Sentilles, P. Štěpán, J. Carlson, and I. Crnković, "Integration of Extra-Functional Properties in Component Models," in *12th International Symposium on Component Based Software Engineering*. Springer, 2009.

[11] I. Kurtev, "State of the art of QVT: A model transformation language standard," in *Applications of Graph Transformations with Industrial Relevance*, ser. Lecture Notes in Computer Science. Springer Berlin, 2008, vol. 5088, pp. 377–393.

[12] T. Leveque, J. Carlson, S. Sentilles, and E. Borde, "Flexible semantic-preserving flattening of hierarchical component models," in *37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE Computer Society, August 2011.

[13] J. Jürjens, "Umlsec: Extending uml for secure systems development," in *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*. London, UK: Springer-Verlag, 2002, pp. 412–425.

[14] T. Lodderstedt, D. A. Basin, and J. Doser, "Secureuml: A uml-based modeling language for model-driven security," in *Proceedings of the 5th International Conference on The Unified Modeling Language*, ser. UML '02. London, UK: Springer-Verlag, 2002, pp. 426–441.

[15] M. Saadatmand, A. Cicchetti, and M. Sjödin, "On the need for extending marte with security concepts," in *International Workshop on Model Based Engineering for Embedded Systems Design (M-BED 2011)*, March 2011.

[16] MARTE specification version 1.0 (formal/2009-11-02), http://www.omgmarte.org.

[17] J. Lloyd and J. Jürjens, "Security analysis of a biometric authentication system using umlsec and jml," in *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 77–91.

[18] S. Chollet, P. Lalanda, and G. Pedraza, "Secure Integration of Service-Oriented Application," September 2009.

[19] M. Jensen and S. Feja, "A security modeling approach for web-service-based business processes," in *Proceedings of the 2009 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, ser. ECBS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 340–347.

[20] J. Lee, K. Kapitanova, and S. H. Son, "The price of security in wireless sensor networks," *Journal of Computer Networks*, vol. 54, pp. 2967–2978, December 2010.