Managing Timing Implications of Security Aspects in Model-Driven Development of Real-Time Embedded Systems

Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sjödin
Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University, Västerås, Sweden
{mehrdad.saadatmand, antonio.cicchetti, mikael.sjodin}@mdh.se

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

Abstract—**Considering security as an afterthought and adding security aspects to a system late in the development process has now been realized to be an inefficient and bad approach to security. The trend is to bring security considerations as early as possible in the design of systems. This is especially critical in certain domains such as real-time embedded systems. Due to different constraints and resource limitations that these systems have, the costs and implications of security features should be carefully evaluated in order to find appropriate ones which respect the constraints of the system. Model-Driven Development (MDD) and Component-Based Development (CBD) are two software engineering disciplines which help to cope with the increasing complexity of real-time embedded systems. While CBD enables the reuse of functionality and analysis results by building systems out of already existing components, MDD helps to increase the abstraction level, perform analysis at earlier phases of development, and also promotes automatic code generation. By using these approaches and including security aspects in the design models, it becomes possible to consider security from early phases of development and also identify the implications of security features. Timing issues are one of the most important factors for successful design of real-time embedded systems. In this paper, we provide an approach using MDD and CBD methods to make it easier for system designers to include security aspects in the design of systems and identify and manage their timing implications and costs. Among different security mechanisms to satisfy security requirements, our focus in this paper is mainly on using encryption and decryption algorithms and consideration of their timing costs to design secure systems.**

Index Terms*—*Real-Time Embedded Systems; Security; Model-Driven Development; Component-Based Development; Runtime Adaptation; Encryption.

I. Introduction

To cope with the specific challenges of designing security for real-time embedded systems, appropriate design methods are required. Due to resource constraints in these systems, the implications of introducing security and its impacts on other aspects and properties of the system should be carefully identified as early as possible and the methods used for designing these systems should provide such a feature [1]. Timing properties are of utmost importance in real-time embedded systems. In this paper, we introduce an approach using Model-Driven and Component-Based Development (MDD & CBD) methods for designing secure embedded systems to bring security aspects into early phases of the development and take into account their timing costs and implications.

This work provides an implementation and a methodology for the generic idea that we discussed in [1] and extends it with the result of our works in [2], [3]. In this work we provide a more complete approach and methodology, compared to the two aforementioned works, based on their combination and synergy and discuss how this approach can cover more issue regarding the timing implications of security mechanisms in real-time embedded systems.

The approach basically works by identifying and annotating sensitive data in the component model of the system, and then deriving automatically a new component model which includes necessary security components for the protection of the data. Our main focus in this paper will be on using encryption and decryption algorithms as security mechanisms. The derivation of the new component model is based on a set of pre-defined strategies. Each strategy defines a different set of possible encryption and decryption algorithms to be used as the implementation of the security components. In this approach, since the derived component model conforms to the original meta model, the same timing analysis and synthesis as for the original component model can be used and applied for the derived one.

With the increasing role of computer systems in our daily lives, there is hardly any software product developed these days that does not have to deal with security aspects and protect itself from malicious adversaries [4]. Also with the exponentially growing number of connected and networked devices and more integration between different tools and services that store and exchange different types of data, not only new types of attacks are constantly emerging but also the risks and consequences of security breaches have become more drastic. Even some simple software products and applications which do not store any sensitive information and therefore may not seem to need to care about security aspects can, for example, be the target of buffer overflow attacks [5] and thus help attackers in gaining access to a system. All these points emphasize that security aspects cannot be taken into account just as an afterthought and added feature to an already developed system [6], but instead should be considered at different phases of development from early phases such as requirements engineering to deployment [4]. What is needed is that instead of adding security features in an “eggshell approach”, security should be designed intrinsically and inseparable from the application to be able to address the threats that target the application itself [6].

Considering security from early phases of development is especially critical in the design of real-time embedded systems. These systems typically have limited amount of resources (e.g., in terms of available memory, CPU and computational power, energy and battery) and therefore, implications of security features should also be taken into account. This is basically because of the fact that Non-Functional Requirements (NFRs), such as security, are not independent and cannot be considered in isolation and satisfying one can affect the satisfaction of others [7]. Therefore, costs and implications of security features should be identified to analyze the trade-offs and establish balance among different non-functional requirements of the system. Such costs can be in the form of impacts on timing, schedulability and responsiveness of the system, as well as memory usage, energy consumption, etc. In real-time embedded systems, satisfaction of timing requirements is critical for the successful behavior of the system, therefore, choice of security mechanisms should be done considering their timing characteristics and impacts.

Model-driven development is a promising approach to cope with the design complexity of real-time embedded systems. It helps to raise the abstraction level, enables analysis at earlier phases of development and automatic generation of code [8], [9]. Component-based development, on the other hand, is another discipline in software engineering and a software development method in which systems are built out of already existing components as opposed to building them from scratch [10], [11]. In other words, it promotes developing a system as an assembly of components by reusing already existing software units (components). Model-driven development and component-based development approaches can be used orthogonally to complement and reinforce each other to alleviate the design complexity of real-time embedded systems [10].

In this context, including security aspects in design models helps with achieving the two goals mentioned so far: bringing security aspects into earlier phases of development and enabling analysis of security implications. Moreover, model-based security analysis (not the focus of this paper) in order to identify possible violations of security requirements [12] becomes possible and also system designers with lower levels of expertise and knowledge in security domain can also include and express security concerns [2]. The latter is due to the fact that code level implementation of security features requires detailed security knowledge and expertise, while at the model level, system designers can use modeling concepts and annotations for expressing security concerns (which in turn may also be used for automatic generation of security implementations).

By constructing the model of the system including security features, timing analysis can then be done on the model to evaluate whether the model meets the timing requirements or not. If so, the implementation of the system can then be generated from the model(s). This leads to a fixed set of security mechanisms that are already analyzed as part of the whole system in terms of their timing behaviors and are thus known to operate within the timing constraints of the system. However, there are situations where such a guarantee in terms of timing behaviors cannot be achieved. For example, in performing analysis some assumptions are taken into account, such as worst-case execution times of tasks. If these assumptions are violated at runtime, the analysis results will not hold anymore. Moreover, in complex real-time systems where timing analysis is not practical/economical or not much information about the timing characteristics of each individual task is available, other approaches are needed in order to tackle the timing issues [13]. One solution is to have runtime adaptation to mitigate timing violations and keep the execution of tasks within their allowed time budgets.

The remainder of the paper is structured as follows. In Section II, we discuss the issue of security and its challenges in embedded systems in general. In Section III, the automatic payment system is described as the motivating example of this paper and also as an example of distributed real-time embedded systems with security requirements. The suggested approach and its implementation are described in Sections IV and V. In Section VI, we introduce a runtime adaptation mechanism to mitigate the violations of timing constraints at runtime. Practical aspects of the introduced approach and other related issued are discussed in Section VII. Section VIII discusses the related work and finally in Section IX conclusions are made.

II. Security in Embedded Systems

In the design of embedded systems, security aspects have often been neglected [14]. However, the use of embedded systems in critical applications such as aviation systems, 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 [15] for the 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 that might be less relevant for other systems such as a database server inside a bank which is physically isolated and protected, in contrast to smart cards and wireless sensor networks which are physically exposed. Physical and side channel attacks [16] are examples of these types of security issues in embedded systems that bring along with themselves requirements on hardware design and for making systems tamper-resistant. Examples of side channels attack could be the use of time and power measurements and analysis to determine security keys and types of used security algorithms.

Increase in the use and development of 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 networked environment.

Besides physical and side channel attacks, often mobility and ease of access of these devices also incur additional security issues. For example, sensitive information other than user data, such as proprietary algorithms of companies, operating systems and firmwares, are also carried around with these devices and need protection.

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 requirements and properties 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. From this perspective, there are many studies that discuss the implications of security features in embedded systems such as [16], in which considering the characteristics of embedded systems, major impacts of security features are identified to be on the following aspects:

- Performance: Security protocols and mechanisms incur heavy computational demands on a system that the processing capacity of an embedded system might not be able to satisfy easily. For example, using encryption and decryption algorithms not only have high computational complexity but also require good amount of memory. In systems that need to handle heavy input loads, such as routers and many systems that are used in telecommunication domain to handle calls and data traffics, these security features can consume lots of processing capacity of the system and result in missed deadlines of other tasks, dropped throughput level, and overall transaction and data rate of the system.
- Power Consumption: In embedded systems with limited power sources, any resource-consuming feature impacts the operational life of the system. In this regard, security features with their heavy computational and memory demands, as discussed above, require careful considerations. There are studies that investigate this issue and compare power consumption of different encryption/decryption algorithms such as [17] that looks at this issue in wireless sensor networks. The issue of power consumption is especially interesting knowing that the growth of battery capacities are a lot slower and far behind the ever-increasing power requirements of security features. This has also led to investigating optimized security protocols for embedded systems and hardware security solutions [16].
- Flexibility and Maintainability: Flexibility of security features and possibility to adapt them according to new requirements is also a challenge in embedded systems. For example, embedded devices such as mobile phones that are used in different operational modes and environments need to support a variety of security protocols. Moreover, security solutions need to be updated in order to be protected against emerging hacking methods. Therefore, flexibility of security design decisions is important for maintaining the security of the system to apply updates and patches.
- Cost: Cost is also a limiting factor in the design of embedded systems. Considering the issues mentioned above, using a faster and more expensive CPU or adding more memory modules to cope with the demands of security requirements can add to the total cost of an embedded system. Taking into account that these devices are often produced in large amounts (e.g. mobile phones and vehicular systems), a small increase in cost can affect overall revenues and competitive potentials of a product in the market. Therefore, the security features that are implemented in embedded systems should be balanced with hardware requirements and consequently cost limits.

III. Motivation Example: Automatic Payment System

Figure 1 shows the Automatic Payment system which is an example of distributed embedded systems with real-time and security requirements. The main goal in the design of this system is to allow a smoother traffic flow and reduce waiting times at tolling stations (as well as parkings).

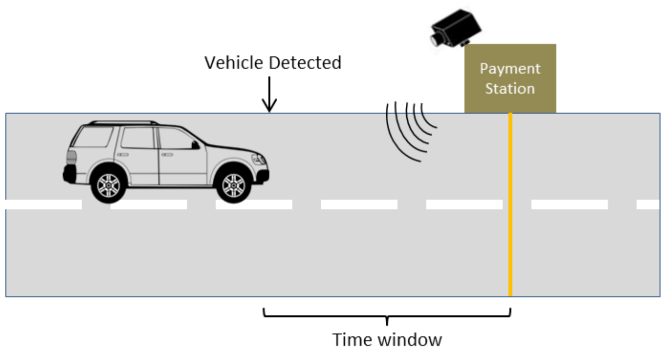


Fig. 1. Automatic Payment System for Toll Roads.

For each toll station, a camera is used to detect a vehicle that 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. This amount is shown to the driver in the vehicle 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 it through a (third party) merchant (e.g., via 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 accordingly. The interactions between different objects in this system are shown in Figure 2.

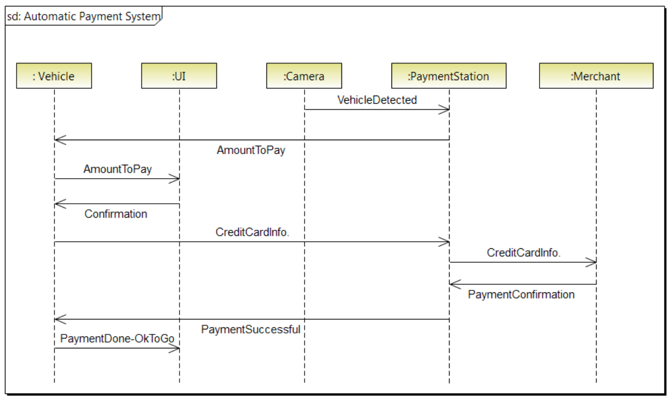


Fig. 2. Automatic Payment System.

To allow a smooth traffic flow, all these operations should be done in a certain time limit. Such time constraints can be calculated considering the specifications of camera and its required time for the detection of an approaching vehicle, 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.

Another scenario that can happen in this system is that several vehicles may approach one station with a short time distance between each which can result in bursty processing loads on the system (analogous to bursty arrivals of aperiodic tasks in real-time terms). In such situations, timing requirements may be violated as even by using static analysis of the system, only certain levels of such bursty loads may be covered and not all the possible cases. One solution to mitigate timing violations in this scenario is to introduce runtime adaptation and adapt the security level of the system at runtime; meaning that security mechanisms that are less time-consuming (and presumably less strong) can be used when such situations are detected. As the last resort, when the system realizes that many number of timing violations are occurring, to maintain a smooth traffic flow and prevent any possible accidents and safety issues due to the increasing queuing of the cars at the tolling station, instead of the on-site payment and charging of the vehicles, the system can just store their information to send a bill later to the owner of the vehicle, or even add the amount to the payment done at the next tolling stations on the road (if there are any and they are connected).

To model and build the system (software parts), particularly considering the timing constraints of the system, the following challenges are identified:

1) Modeling security mechanisms with enough details to enable both timing analysis on the model and synthesis of the security implementations,
2) Obtaining timing costs of security mechanisms,
3) Managing possible timing violations of the system at run-time.

The first challenge is discussed in the following two sections. To get the timing costs of security mechanisms, we rely on studies such as [18] that have done such measurements. To solve the third challenge, a runtime adaptation mechanism is introduced and we show how it helps to mitigate the runtime violations of timing constraints.

IV. Approach

Based on the identified challenges in the previous example, we introduce an approach that aims to bring the security concerns in the design of embedded systems. Our suggested approach helps systems designers in expressing the security concerns in a system without the need to have much security expertise on the actual implementation of security mechanisms. It does so by just requiring the system designers to identify sensitive data entities that need to be protected. In the scope of our work, it can be for example the data that need to be confidential and/or whose sender must be authenticated. Moreover, to mitigate potential timing violations of security mechanisms at runtime, the approach provides the option to include an adaptation feature for the security mechanisms.

To implement the approach, ProCom [19] component model has been used; although the approach is not dependent on this specific component model and can be implemented using other component models as well. Security needs are specified as annotations on the component model. A benefit of the ProCom component model is its power in defining new attribute types using its attribute framework to annotate and specify new types of data. The term component model hereafter is used to basically refer to the component architecture model than the meta-model of ProCom. From the specification of the security needs at the data level and physical platform level, a model transformation is applied on the component model to derive a new component model including security implementations. The derivation of the new component model (which now has appropriate security components implementing the security needs) is done based on a selected strategy. The strategy basically specifies the preferences in terms of security implementations and which of them to choose among a set of different possible ones. Having the necessary information in the model, the steps that have been described so far can be summarized as follows:

1) The component model which specifies the functional and non-functional (extra-functional) part of the system is transformed into a functionally equivalent model with added security implementations;
2) Analysis can be performed on the derived component model that includes security components to identify any possible violations of timing constrains; and
3) Finally, the system is synthesized.

The considered process is iterative and allows to refine security specification after evaluating the resulted system properties such as timing properties. In other words, timing analysis, for example, can be performed on the derived component model and if timing properties of the derived model do not satisfy the timing requirements, the derivation process can be repeated with different preferences to finally gain a model which is satisfactory in terms of timing requirements. The process is depicted in Figure 3 showing different models and annotations that are used as input in each step (i.e., the analysis of the system model as well as synthesis of the implementation).

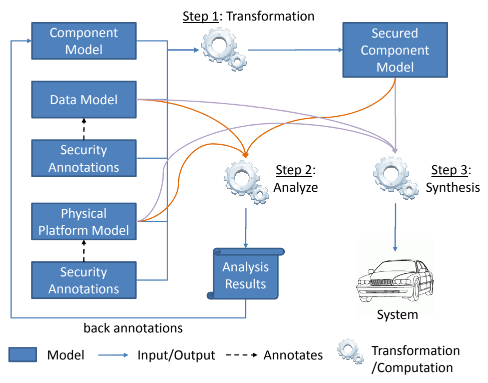


Fig. 3. General description of the approach process.

V. Implementation

A. ProCom Component Model

While the approach principles are 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.

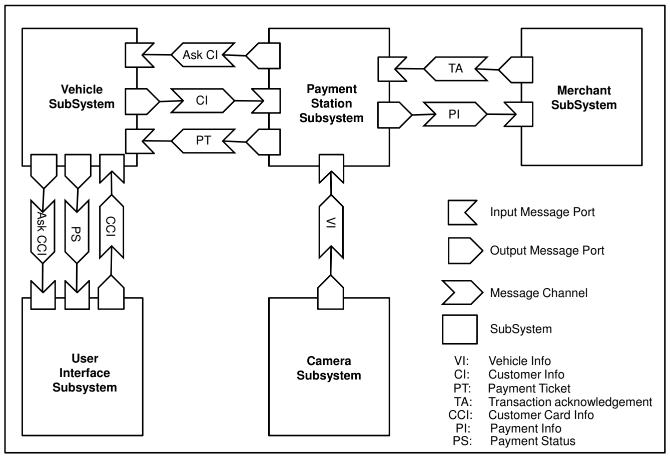


Fig. 4. Component Model of the System using ProCom.

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.

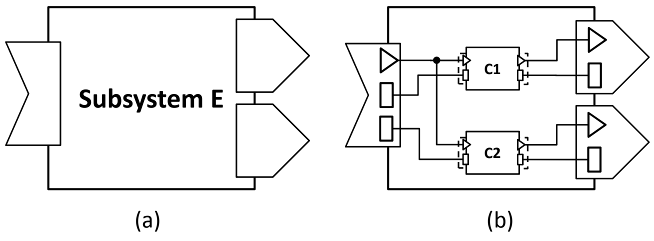


Fig. 5. ProCom SubSystem Implementation.

B. Data Model

As components are usually intended to be reused, their related data may also be reused. To this end, we propose to extend the data-entity approach described in [20] for design-time management of data in component-based real-time embedded systems. In this approach every data entity is stored in a shared repository and designers are provided with an additional architectural view for data management, namely the data architectural view. The description of a data entity 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. We use the concept of data entity to identify data that are transfered through different message channels in the system (shown in Figure 4) and map them to their respective security concerns (e.g., if they need to be encrypted and protected or not). Table I and Table II show the data entities in 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 the 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 the 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

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. 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 Units (ECUs), called Physical Nodes, including their configurations such as processor type and frequency, the connections between the physical nodes, and the physical platforms which represent a set of ECUs fixed together.

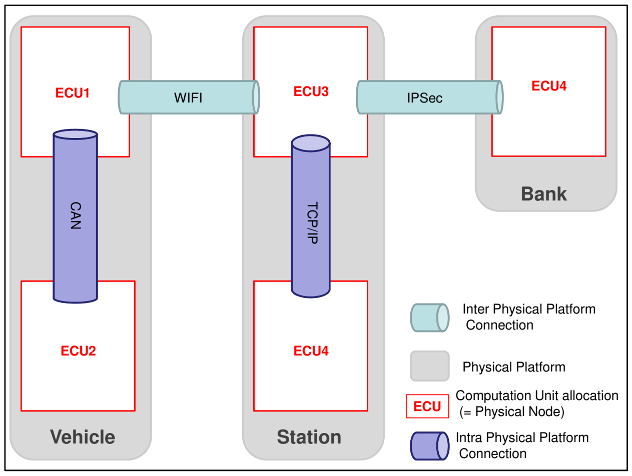

ECU1
WIFI
ECU3
IPSec
ECU4
CAN
TCP/IP
Bank
ECU2
ECU4
Vehicle
Station
Inter Physical Platform Connection
Physical Platform
ECU
Computation Unit allocation (= Physical Node)
Intra Physical Platform Connection

Fig. 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 a one to one mapping between virtual node and physical node.

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

- **Confidentiality** ensures that the considered information cannot be read by any external person of the system; and

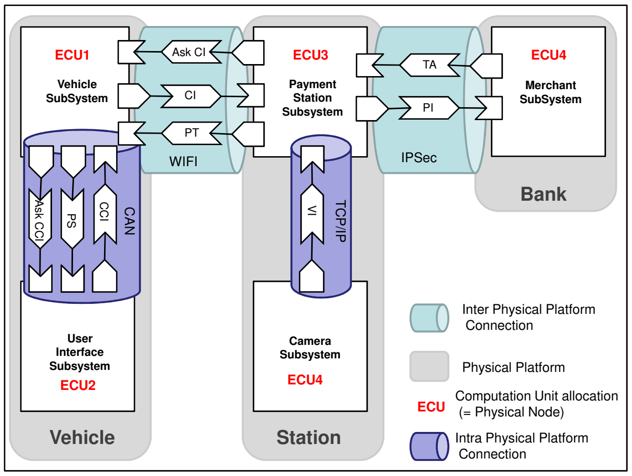


Fig. 7. Deployment Model of the System depicting allocation to Physical Platforms.

- **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:

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

Using these properties, the person responsible for 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 (by construction). 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

E. Cost of Security Implementations

Different encryption/decryption algorithms as security mechanisms can be selected to satisfy the identified security properties in the system. Considering the fact that each security mechanism in the system has its own costs in terms of timing and performance, power consumption and so on, choosing an appropriate security mechanism is critical in order to ensure the satisfaction of timing requirements of the system. For this purpose, and to take into account the timing costs of different security mechanisms, we rely on the results of studies such as [18] 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 times can be computed knowing the target platform, algorithm, key size and data size. A timing estimation toolkit may also be provided which provides execution time estimates based on these measurements. As can be observed from the table, we also take into account and add this flexibility that some algorithms may not be supported on some platforms (marked as NS).

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)

F. 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 among all possible security implementations:

- The **StrongestSecurity** strategy selects the strongest security implementation available on the platforms (taking into account that some security mechanisms, namely encryption algorithms here, may not be available and possible on a certain platform, hence selecting the strongest available one);
- The **StrongestSecurityAndLimitImplemNb** strategy selects the strongest security implementation available on the platforms while ensuring that we use as few as possible different security implementations, since each message channel can use a different encryption algorithm (finding the most common security implementation which achieves the strongest level in terms of the strength rankings);
- 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 while ensuring 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.

G. *Transformation*

The transformation is performed in four steps:

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

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 needs to be confidential. The pseudo algorithm of the transformation is described in Listing 1.

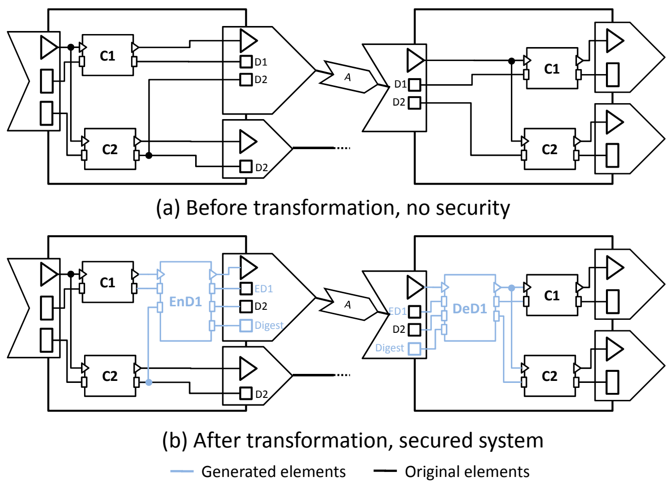

C1
C2
D1
D2
A
(a) Before transformation, no security
EnD1
DeD1
ED1
Digest
(b) After transformation, secured system
Generated elements
Original elements

Fig. 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 the digest used for authentication. The decryption component (DeD1) ensures that all message data will be available at the same time through the 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.

VI. Runtime Adaptation

The suggested approach results in a *static* and fixed set of security mechanisms to be implemented and used in each invocation and use of the system. The system model including the added security components can then be analyzed in terms of timing properties before reaching the implementation phase and therefore it can be evaluated whether the timing requirements are met or not.

There are, however, cases where such static analysis may not be possible or even economical. For example, when there is not much timing information available about each task in the system to perform timing analysis, particularly in complex real-time systems with a big number of different tasks. In such systems even if enough timing information is available for each task, due to the complexity and big number of tasks, performing timing analysis may actually be not economical. Moreover, in performing static analysis some assumptions are taken into account and if those assumptions are violated at runtime then the static analysis results will not hold anymore. In such situations, a runtime adaptation mechanism can help to cope with the above challenges and mitigate timing violations by establishing balance between timing and security in a *dynamic* fashion.

To bring such adaptation mechanism into our approach, we introduce another strategy called **StrongestSecurityAdaptive**. By selecting this strategy, the implementation of added security components will be synthesized as depicted in Figure 9.

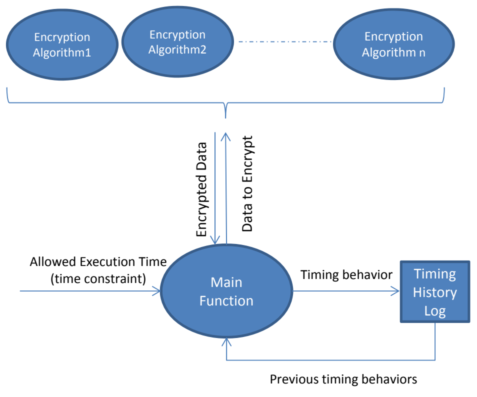


Fig. 9. Adaptation mechanism.

As shown in Figure 9, by using this strategy, in the body of the added security components (here encryption ones), the implementation of all different possible encryption algorithms will also be included. When a request for an encryption arrives, the component firstly tries to use the strongest possible encryption algorithm (based on the rank of algorithms in Table VI) to encrypt the data. The time it takes to perform the encryption is stored in the `Timing History Log`. If this time is more than the specified timing constraint for performing the job, then for the next encryption, another encryption algorithm with a lower rank but with less execution time will be selected (based on the information in Table VI).

In case the encryption job completes sooner than the specified time limit, the unused portion of its time budget is then used to determine whether it is feasible to adopt a higher ranked algorithm for the next encryption job or not. With this approach, the feedback that is produced regarding the timing behavior of encryption algorithm is used by the system to try to adapt itself. Therefore, when the system receives a burst of processing loads which it cannot fulfill under specified time constraints, it adapts itself to this higher load and similarly when the processing load decreases, it can gradually go back to using more time-consuming (and presumably more secure) encryption algorithms. This design is based on the implicit assumption that when it is detected that an executing encryption algorithm is exceeding its allowed time budget, it is basically more costly to terminate it in the middle of the encryption procedure, and restart the encryption of the data with another encryption algorithm, than just letting it finish its job, and instead use one with a lower execution time in the next invocation of encryption components.

The information that is logged in the `Timing History Log` has the following format: *Timestamp, Encryption algorithm, Time constraint, Actual execution time* (timestamp, time constraint and actual execution time are in system ticks unit in the following experiment). An example of the generated log information is shown in Table VII.

TABLE VII
Sample log information.

```
10360, AES, 50, 90
11800, 3DES, 80, 70
14500, 3DES, 60, 70
21353, DES, 60, 10
22464, 3DES, 90, 40
23112, AES, 50, 50
28374, AES, 60, 58
```

Considering the last row from the log as:

ts, alg, t, e

(ts: timestamp, alg: encryption algorithm, t: time constraint, e: actual execution time)

the decision that the system should adopt a lower ranked algorithm is made using the following formula:

(i) $e > t \Rightarrow$ *move down in the encryption algorithms table and select the next algorithm with a lower rank.*

Also, considering the two log records described as follows:

ts(l), alg(l), t(l), e(l) : representing the last log record

ts(h), alg(h), t(h), e(h) : representing the log record for the first encryption algorithm with a higher rank that was used before the last log record;

the decision to adopt a higher ranked algorithm is made using the following formula:

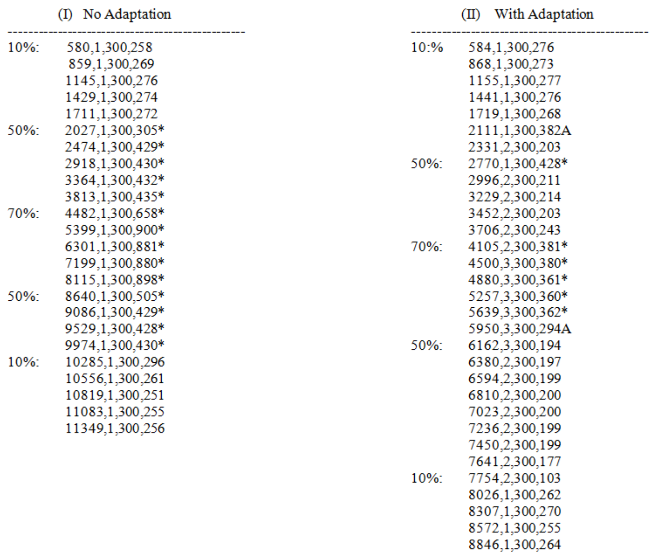
(ii) $e(l) < t(l) \wedge t(l) - e(l) > abs(e(h) - t(h)) \Rightarrow$ *move up in the encryption algorithms table and select the previous*

higher ranked algorithm.

A. Evaluation of the Adaptation Mechanism

In [3], we have tested the introduced adaptation mechanism; here we include the evaluation results produced during that work to demonstrate the benefits of using the adaptation approach. A simulation environment was setup as described in [3] with the use of a tool called CPU Killer [21] to enforce arbitrary CPU loads at desired times.

Figure 10 shows the evaluation results comparing performing encryption with and without using the adaptation mechanism. In each case, CPU loads of 10%, 50%, 70%, and then back to 50%, and 10% were applied.



```
        (I)  No Adaptation
---------------------------------------------
10%:     580,1,300,258
          859,1,300,269
         1145,1,300,276
         1429,1,300,274
         1711,1,300,272
50%:     2027,1,300,305*
         2474,1,300,429*
         2918,1,300,430*
         3364,1,300,432*
         3813,1,300,435*
70%:     4482,1,300,658*
         5399,1,300,900*
         6301,1,300,881*
         7199,1,300,880*
         8115,1,300,898*
50%:     8640,1,300,505*
         9086,1,300,429*
         9529,1,300,428*
         9974,1,300,430*
10%:     10285,1,300,296
         10556,1,300,261
         10819,1,300,251
         11083,1,300,255
         11349,1,300,256
```

```
        (II)  With Adaptation
---------------------------------------------
10:%     584,1,300,276
         868,1,300,273
         1155,1,300,277
         1441,1,300,276
         1719,1,300,268
         2111,1,300,382A
         2331,2,300,203
50%:     2770,1,300,428*
         2996,2,300,211
         3229,2,300,214
         3452,2,300,203
         3706,2,300,243
70%:     4105,2,300,381*
         4500,3,300,380*
         4880,3,300,361*
         5257,3,300,360*
         5639,3,300,362*
         5950,3,300,294A
50%:     6162,3,300,194
         6380,2,300,197
         6594,2,300,199
         6810,2,300,200
         7023,2,300,200
         7236,2,300,199
         7450,2,300,199
         7641,2,300,177
10%:     7754,2,300,103
         8026,1,300,262
         8307,1,300,270
         8572,1,300,255
         8846,1,300,264
```

Fig. 10. Performing encryption with and without adaptation.

The columns for each log record in Figure 10 identify: system time (ticks), encryption algorithm (AES=1, 3DES=2, DES=3) , time constraint (for each invocation; in ticks), and actual execution time (ticks). The records in which the violation of the time constraint has occurred are marked with a '*'. Comparing the two cases (without adaptation and with it) shows that the number of time constraint violations are reduced in the second case compared to the first case where only one encryption algorithm (with the highest execution time) is used. Moreover, in the second case more number of encryption jobs have been performed under a shorter period of time.

Since the goal with this adaptive strategy is to use the strongest security algorithm possible, the adaptation mechanism assumes that the encryption algorithms in Table VI are sorted according to their execution times resulting in the strongest but most time consuming one to be at the top and the weakest but less timing consuming algorithm at the bottom. Also, as a note for the decryption side, there are different ways to match and synchronize the decryption algorithm with the selected encryption algorithm. Our suggested way to do this is to add some additional bits identifying the used encryption algorithm (e.g., through the use of 2-bit or 3-bit ID numbers, according to the number of different algorithms) to the encrypted message for the decryption side to correctly pick and use the appropriate algorithm. Moreover, in the introduced adaptation mechanism and its evaluation, an encryption algorithm and the respective decryption algorithm for it have been assumed to take the same amount of time which is generally valid as mentioned in [18]. However, to extend the adaptation approach for distributed systems where encryption and decryption can be performed on different nodes, more parameters for making adaptation decisions can be added. Such an extension can be to consider the sum of encryption time and decryption time for each algorithm to make adaptation decisions instead of just considering the encryption time only.

VII. Discussion

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 [22] which allows to introduce additional attribute to any model element in ProCom. The model transformation has been implemented using a QVTo [23] transformation plugged at the end of the process described in [24]. These experiments aim to show the benefits at the 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 StrongestSecuritySchedulable strategy is hard to implement, however, it is the most interesting one. One of the reasons that we do not claim that we also provide this strategy, in spite of having the execution times of security components, is that the actual execution times in the synthesized system will not necessarily be the sum and individual addition of the execution times of the added security components to the rest of the system. More complex security implementation strategies can be considered but are not covered in this paper.

As for the synthesis of the code of the security components, in order to keep the approach generic, we intend to let certificate specification and other specific parameters of encryption algorithm to be filled in the generated code. One generator is associated for each algorithm. The suitability for timing analysis of the generated component code needs 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 also to react to authentication errors. This problem could be partially solved by allowing developers to add code to manage authentication errors in the generated code to define what must be the output data in each specific case.

Regarding the runtime adaptation mechanism, while on one hand, it may make the job of the attackers harder as not a fixed algorithm is used in each invocation and thus it will not be known and predictable to the attackers (hence some sort of "security through obscurity"), on the other hand, if attackers know the internal mechanism of the runtime adaptation, they can force some processing load on the system to make the system adopt the weakest algorithm possible, and that way, make it easier for themselves to break into the system. Moreover, the adaptation mechanism which was used as part of our general approach in this paper, can also be designed to act as an option; in the sense that it can be turned on and used when a processing load beyond a certain level is detected and turned off otherwise. This can help to mitigate the overhead of the adaptation mechanism itself (although another mechanism to monitor the processing load would need to be added in that case) and only use it when there are many requests for encryption.

VIII. Related Work

Designing security features for real-time embedded systems is a challenging task and requires appropriate methods and considerations. [16] and [25] particularly discuss the specific challenges of security in embedded systems and define it as a new dimension to be considered throughout the development process. Considering the unique challenges of security in embedded systems, [25] also emphasizes that new approaches to security are required to cover all aspects of embedded systems design from architecture to implementation. The methods that we introduced in this paper contribute towards this goal by applying different disciplines in the field of software engineering, such as model-driven development methods, to cope with the specific challenges of designing security for embedded systems.

Also as a non-functional requirement [7], [26], satisfying security requirements in a system has costs and implications in terms of impact on other requirements such as performance, power consumption and so on. In [17], measurement and comparison of memory usage and energy consumption of several encryption algorithms on two specific wireless sensor network platforms have been done. Performance and timing comparisons of several encryption algorithms are offered in [18] where Pentium machines are used as the platform. The approaches we proposed in this paper, work by relying on the timing and performance comparison results of encryption algorithms in such studies.

While model-driven and component-based approaches serve as promising approaches to cope with the design complexity of real-time embedded systems, management of runtime data in these systems has also become an important issues than ever before due to the growing complexity of them. This fact becomes more clear when we realize that keeping track of all data that are passing through different parts of the system is an extremely hard task for a person. In addition, most design methods based on component models focus mainly on functional structuring of the system without considering semantics and meanings for data flows [20]. A data-centric approach for modeling data as well as using real-time databases for runtime data management in real-time embedded systems is proposed in [20]. In this work, however, non-functional (extra-functional) properties such as security are not addressed, and our approach presented in this paper basically follows a similar method for modeling data entities as a basis to define security specification.

As for modeling security aspects, there are several solutions such as UMLsec [12]. UMLsec is a UML profile [27] for the specification of security relevant information in UML diagrams. It is one of the major works in this area and comes with a tool suite which provides the possibility to evaluate security requirements and their violations. SecureUML [28] is also another UML profile for modeling of role-based access controls. UML profile for Modeling and Analysis of Real-time Embedded Systems (MARTE) [29] provides semantics for modeling non-functional properties and their analysis (e.g., schedulability). In [30], we have discussed MARTE and the benefits of extending MARTE with security annotations to better cover the modeling needs of embedded systems. Besides UMLsec and its tool suite which enables analysis of security requirements, in [31], a method for specifying security requirements on UML models and verifying their satisfaction by relating model-level requirements to code-level implementation is offered. In [32], we have provided a small example how it is possible to model security requirements along with some other requirements of telecommunication systems and then perform model-based analysis using the analysis tool suite of UMLsec to identify possible violations of security requirements.

The need to identify sensitive data is also discussed in [33] where an extension to include security concerns as a separate model view for web-services based on Web-Services Business Process Execution Language (WS-BPEL) is offered. However, it does not take into account the consequences of security design decisions on timing aspects, while by identifying sensitive parts of messages which need to be secured, our objective is to ease the computation of the timing impacts of security implementations protecting those sensitive data. Considering the challenges of securing distributed systems [34] has done a survey on the application of security patterns, as a form of software design patterns, to secure distributed systems. Moreover, it discusses different methodologies that make use of these ad-hoc security patterns in a structured way. It also reports that the majority of the studied methodologies lack explicit support for distributed systems and special concerns that these systems have and mentions the development of tailored methodologies for different types of distributed systems as an important future work in this area. The approach that we suggested here could serve as an example for developing such methodologies in particular for distributed real-time and embedded systems in which timing requirements play a key role in the correctness of the whole system.

Regarding the adaptation method that we used as part of our suggested approach, there are also several related studies and approaches that we discuss them here. The study done in [35] is one of the interesting works in the area of security for real-time embedded systems which uses an adaptive method.

In this work, the main focus is on a set of periodic tasks with known real-time parameters, whereas, our main target is complex systems that can consist of any type of real-time tasks. Also, while in our work, the security level of the system is considered implicitly through the selection of algorithms from the encryption algorithms table, in [35], a QoS value has been considered which explicitly represents the security level of the system. Moreover, in our work, it is the encryption algorithms which are adaptively replaced, while the main adaptation component in that work is the key length. Our approach can easily be extended to cover not only different encryption algorithms but also variations of each, including different combinations of key length, number of rounds and so on, as items (rows) in the encryption algorithms table (e.g., AES256, AES128, etc.). Another interesting study with is close to our work is [36], which basically introduces a similar type of adaptation mechanism as ours. The main focus in that work is, however, on client-server scenarios using a database, and to manage the performance of transactions. The security manager component used int his work periodically adjusts the the security level of the system. In our approach, however, the adaptation mechanism is executed per request and is not active when there is no request for encryption. Moreover, it is possible in the approach introduced in this work that an inappropriate encryption method is used by a client, while security level change is occurring. To solve this situation, several acknowledgment messages are sent and the process is repeated to correct this issue. Therefore, it is possible that the security manager faces problems regarding synchronization and message loss due to out of order arrival of messages. As another approach for managing security in real-time systems, in [37], a secure-aware scheduler is introduced which basically incorporates and takes into account timing management of security mechanisms as part of its scheduling policy.

IX. Conclusion and Future Work

In this paper, we introduced an approach to define security specifications in real-time embedded systems at a high level of abstraction based on the benefits of model-driven and component-based methods. Using the suggested approach we bring semantics to the data that are transferred in embedded systems to identify sensitive data. The approach enables also to derive automatically the security implementations and facilitates performing timing analysis including security features at early phases of development. It was also demonstrated how incorporating a runtime adaptation mechanism as part of the approach helps to mitigate the violations of timing constraints at runtime. As mentioned, such runtime adaptation mechanisms are especially useful for complex systems where performing static analysis may not be practical, as well as in cases where the assumptions that have been used for performing static analysis are prone to deviation and violation at runtime which can then lead to the invalidation of analysis results. Moreover, the introduced approach helps system designers to mainly focus on the system architecture and addressing timing properties, and at the same, including security concerns in the design models without needing much expertise on how to implement security mechanisms. This again contributes to bringing security considerations in higher levels of abstraction.

One of the extensions of this work is to define and add more strategies for the designers to choose. Among the currently defined strategies, the StrongestSecuritySchedulable is the most interesting one but is hard to implement and will be part of our future works. One of the reasons that we do not claim that we also provide this strategy, in spite of having the execution times of security components, is that the actual execution times in the synthesized system will not necessarily be the sum and individual addition of the execution times of the added security components to the rest of the system. Also as another idea for the extension of this work, it would be interesting to define and assign required security strength to data and message channels as another factor that also affects the selection of security components. It should also be noted that in this work we mainly addressed encryption as a security mechanism. Considering other mechanisms such as authorization methods and their impacts on timing characteristics of systems is another interesting direction of this work to investigate. Also including other aspects than timing, such as power consumption of security mechanisms, performing trade-off, and establishing balance among them, similar to what we did here for timing properties, can be another extension of this paper and future work.

X. Acknowledgements

This work has been supported by Xdin Stockholm AB [38] and Swedish Knowledge Foundation (KKS) [39] through the ITS-EASY industrial research school program [40] .

References

[1] M. Saadatmand, A. Cicchetti, and M. Sjödin, "On generating security implementations from models of embedded systems," in *The Sixth International Conference on Software Engineering Advances (ICSEA 2011)*, Barcelona, Spain, October 2011.

[2] M. Saadatmand and T. Leveque, "Modeling security aspects in distributed real-time component-based embedded systems," in *Information Technology: New Generations (ITNG), 2012 Ninth International Conference on*, Las Vegas, USA, april 2012, pp. 437 –444.

[3] M. Saadatmand, A. Cicchetti, and M. Sjödin, "Design of adaptive security mechanisms for real-time embedded systems," in *Proceedings of the 4th international conference on Engineering Secure Software and Systems*, ser. ESSoS'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 121–134.

[4] P. T. Devanbu and S. Stubblebine, "Software engineering for security: a roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 227–239.

[5] F.-H. Hsu, F. Guo, and T.-c. Chiueh, "Scalable network-based buffer overflow attack detection," in *Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, ser. ANCS '06. New York, NY, USA: ACM, 2006, pp. 163–172.

[6] A. Main, "Application security: Building in security during the development stage," *Journal of Information Systems Security*, vol. 13, no. 2, pp. 31–37, 2004.

[7] L. M. Cysneiros and J. C. S. do Prado Leite, "Non-functional requirements: From elicitation to conceptual models," in *IEEE Transactions on Software Engineering*, vol. 30, no. 5, 2004, pp. 328–350.

[8] 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.

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

[10] M. Törngren, 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.

[11] I. Crnkovic, "Component-based Software Engineering - New Challenges in Software Development," in *Software Focus*, vol. 2, 2001, pp. 27–33.

[12] B. Best, J. Jurjens, and B. Nuseibeh, "Model-based security engineering of distributed information systems using umlsec," in *Proceedings of the 29th international conference on Software Engineering*, ser. ICSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 581–590.

[13] A. Wall, J. Andersson, J. Neander, C. Norström, and M. Lembke, "Introducing temporal analyzability late in the lifecycle of complex real-time systems," in *Real-Time and Embedded Computing Systems and Applications*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, vol. 2968, pp. 513–528.

[14] 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.

[15] E. Albrechtsen, "Security vs safety," NTNU - Norwegian University of Science and Technology http://www.iot.ntnu.no/users/albrecht/rapporter/notat%20safety%20v%20security.pdf, Accessed: December 2012.

[16] 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, 2004, pp. 753–760, moderator-Ravi, Srivaths.

[17] 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.

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

[19] 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.

[20] 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.

[21] CPU Killer, http://www.cpukiller.com/, Accessed: December 2012.

[22] 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.

[23] 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.

[24] 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.

[25] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in embedded systems: Design challenges," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, pp. 461–491, August 2004. [Online]. Available: http://doi.acm.org/10.1145/1015047.1015049

[26] M. Saadatmand, A. Cicchetti, and M. Sjödin, "Toward model-based trade-off analysis of non-functional requirements," in *38th Euromicro Conference on Software Engineering and Advanced Applications(SEAA)*, September 2012.

[27] B. Selic, "A systematic approach to domain-specific language design using uml," in *Object and Component-Oriented Real-Time Distributed Computing, 2007. ISORC '07. 10th IEEE International Symposium on*, may 2007, pp. 2 –9.

[28] 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.

[29] MARTE specification version 1.1, http://www.omgmarte.org, Accessed: December 2012.

[30] 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.

[31] 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.

[32] M. Saadatmand, A. Cicchetti, and M. Sjödin, "Uml-based modeling of non-functional requirements in telecommunication systems," in *The Sixth International Conference on Software Engineering Advances (ICSEA 2011)*, October 2011.

[33] 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.

[34] "Securing distributed systems using patterns: A survey," *Computers & Security Journal*, vol. 31, no. 5, pp. 681 – 703, 2012.

[35] K.-D. Kang and S. H. Son, "Towards security and qos optimization in real-time embedded systems," in *SIGBED Rev.*, vol. 3. New York, NY, USA: ACM, January 2006, pp. 29–34.

[36] S. H. Son, R. Zimmerman, and J. Hansson, "An adaptable security manager for real-time transactions," in *Proceedings of the 12th Euromicro conference on Real-time systems*, ser. Euromicro-RTS'00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 63–70.

[37] T. Xie and X. Qin, "Scheduling security-critical real-time applications on clusters," in *IEEE Transactions on Computers*, vol. 55, no. 7, july 2006, pp. 864 – 879.

[38] Xdin AB, http://xdin.com/, Accessed: December 2012.

[39] KK-stiftelsen: Swedish Knowledge Foundation, http://www.kk-stiftelsen.org/SitePages/Startsida.aspx, Accessed: December 2012.

[40] ITS-EASY post graduate industrial research school for embedded software and systems, http://www.mrtc.mdh.se/projects/itseasy/, Accessed: December 2012.