SysML-Sec: A SysML environment for the design and development of secure embedded systems

Yves Roudier, Muhammad Sabir Idrees
EURECOM
Network and Security Department
Sophia Antipolis, France
Yves.Roudier@eurecom.fr, Muhammad-Sabir.Idrees@eurecom.fr

Ludovic Apvrille
Institut Mines-Telecom, Telecom ParisTech
CNRS/LTCI, System-On-Chip laboratory (LabSoC)
Sophia Antipolis, France
ludovic.apvrille@telecom-paristech.fr

***Abstract*—Designing a secure system has always been a difficult exercise. In practice, much of the focus for designers and developers has been on delivering a working system in the first place. Security concerns have long been considered only in retrospect, especially after serious flaws are discovered. On the contrary, safety issues are commonly taken into account from the very first development phases. However, the size, heterogeneity, and communication features of modern embedded systems make it compelling to develop an appropriate engineering methodology to more explicitly define security objectives and threats. All this complexity also makes it compelling to verify that requirements are consistent with and satisfied by a candidate design before any commitment to a particular implementation.**

This paper introduces SysML-Sec, a new SysML environment that makes it possible for security experts to intervene at all methodological stages, as well as to assess the impact of security over safety. Safety and security requirements are first captured within extended SysML Requirement diagrams. Attacks are organized within Parametric diagrams, where assets are represented with blocks. Since assets in embedded system are composed of functional and architectural elements of the system, requirements are linked to assets, and subsequently attacks, in a partitioning stage. The partitioning phase includes a functional description, a description of the hardware architecture, and a mapping stage in which functions and communications between functions are to be mapped over hardware components. Once partitioned, software-implemented functions are designed with communicating blocks and state machines. Executable code can then be generated from design diagrams.

An automotive embedded application, developed with industrial partners in the scope of the FP7 European project EVITA, illustrates the relevance of SysML-Sec. This use case was modeled with the open-source tool TTool, which supports SysML-Sec as well as safety/security simulation and formal proof.

***Index Terms*—Communication System Security; Computer Security; Design methodology; System-level design; Component Architectures; Embedded Software**

I. Introduction

ONLY THE ABSTRACT IS TO BE CONSIDERED

Embedded systems are pervading our daily experience of technology as they are now commonplace in vehicles and transportation systems as well as in mobile devices. Command and control designers are also increasingly resorting to embedded systems. Embedded systems are made of both software and hardware electronic components that are tightly coupled together to form the embedded system architecture. The software part of those systems offers flexibility to products after they are released to the market, for instance through firmware updates - something impossible with the hardware. Many of those systems exhibit a heterogeneous and distributed architecture, featuring multiple computational units (microprocessors or microcontrollers) interconnected with communication busses.

A. Security Issues in Embedded Systems.

Security mechanisms now are used to separate areas in a processor whose communication is mediated: this is typically the case of virtualization technologies, or of trusted computing technologies. The latter are best illustrated with the Trusted Computing Group's TPM or ARM's Trusted Zones. Attacks on embedded systems can be physical or logical. For instance, probing attacks targeting the communication bus between a microprocessor and its memory may leak secret keys or protected software code because it is not encrypted: this was for example the case with the attack targeting Microsoft's XBox. Logical attacks exploiting a vulnerability in the implementation of an applications running on top of a micro-controller of the embedded system have now also become possible because many embedded systems have communication interfaces that may receive maliciously crafted data. A successful attack may in turn make it possible to take control of the execution environment and to probe communication interfaces or to inject further malicious traffic onto them. Such vulnerabilities have typically been exploited in jailbreak campaigns against operating systems of mobile phones. Physical attacks may include techniques like on-chip probing with a laser or fault injection through the use of heavy-ion radiations.

B. Security Requirements and Embedded Systems.

Designing a secure system has always been a complex exercise. In practice, much of the focus for designers and developers has been on delivering a working system in the first place. Security concerns have long been considered only in retrospect, especially after serious flaws are discovered. The size and heterogeneity of a modern embedded system, which can be seen as a system of systems, makes it compelling to develop an appropriate security requirements engineering methodology to support the systematic description of the motives for securing a system and appropriate means to do so, for both hardware and software artifacts. This methodology should also support the validation of the requirements expressed. Defining security goals or objectives and acknowledging the existence a malicious environment early can help steer the design of a system architecture into the right direction well before designers and developers are committed to a particular implementation or when time-to-market is of utmost importance. Designers cannot afford to defend against all potential threats, because security mechanisms are costly, and also because they may degrade other qualities of the system, in particular its safety or performance. In the remainder of the paper, we discuss what the requirements analysis should capture, especially with respect to the architecture and design lifecycle, and also with respect to assessing the impact of security on system properties like safety. We introduce a SysML-based iterative process for embedded systems that supports the model-driven identification of security requirements which we organize as a set of goals.

C. Outline of the Paper.

Section 2 introduces some background on the design of embedded system architectures, how they relate to requirement engineering, and why we stick to SysML. This section also introduces a case study we use throughout the paper. Section 3 discusses the nature of security requirements in embedded systems and how to model them in SysML. Section 4 deals with the impact of security requirements over the system architecture, and in particular the hardware/software partitioning of the system. Section 5 summarizes the security requirements definition methodology which we evaluate based on the case study mentioned above. Section 6 compares our contributions with related work. We finally summarize our findings and sketch future work in Section 7.

II. Embedded Systems: Architecture Design and Requirements

A. The Example of Automotive On-Board Systems.

Throughout this paper, we will illustrate our analysis and methodology with results from a study that we conducted in the scope of the FP7 European research project EVITA. The project, which included a major car manufacturer and tier-one equipment suppliers, investigated the security of so-called automotive on-board networks. These are the embedded systems that are currently deployed in modern vehicles and that control in particular the engine, the brakes and ABS or ESP systems, as well as the dashboard console or entertainment systems like the CD player. All microcontrollers used in such systems are termed Electronic Control Units (ECUs). A modern vehicle easily contains a hundred of such ECUs, some being very basic, some other being as powerful as a small personal computer. ECUs are connected together through busses of different technologies (LIN, CAN, Flexray ...) that are organized in different local networks or domains interconnected by gateways.

We present in this paper the result of experiments in applying the security requirement engineering methodology we describe in this paper to the EVITA use cases. Those use cases in particular featured two scenarios that we will use in this paper. The first one deals with maintenance issues at the workshop. It notably addresses the need to update the firmware of an ECU in a secure manner that should protect both the intellectual property of the car maker or equipment supplier and the integrity of the update process. The second use case relates to the introduction of car-to-car communications to improve road safety. It aims at notifying emergency braking events from a car to its immediate followers, thereby making it possible to display a warning sign on the dashboard and speed up or even automate the braking action. A prototype secure automotive on-board network was designed and developed by the EVITA partners, and tested within cars. Security here must be introduced to prevent an attacker from injecting a fake notification into the sending or receiving vehicle. Such attacks are feasible either through the Internet connection increasingly available in vehicles or through the connection of a compromised mobile phone to the vehicle on-board network, a feature now available in more and more vehicles. Further details about these scenarios and security implementations are available in [1] or from the EVITA project web site [1]. We will illustrate our requirements definition process with these examples.

B. Hardware / Software Partitioning

Software-centric systems are commonly designed with a V-cycle comprising the following stages: requirements elicitation, software analysis, software design, implementation. Each of these stages are then followed with a corresponding verification state, that commonly relies on testing, simulation and formal verification techniques. In the case of embedded systems, that approach is obviously applicable only once functions to be software implemented have been specified. In other words, the V-cycle can start only once the software and hardware partitioning has been performed.

System partitioning is a process to analyze various functionally equivalent implementation of systems specifica-

[1]www.evita-project.org/

tion. It usually relies on the Y-chart approach [2] depicted in Figure 1:

1) *Applications* are first described as abstract communicating tasks: tasks represent functions independently from their implementation form. The traditional top down methodology starts with an informal description of a system from which a reference model is first developed. Typical languages for these first models are Matlab, C, or C++. That first model is verified against functional correctness, and might as well be used to drive first performance evaluations.
2) Targeted *architectures* are independently described from tasks. They are usually described with a set of execution nodes (e.g., CPU), communication nodes (e.g., buses), and storage nodes (e.g., memories).
3) A *mapping* process defines how application tasks and abstract communications are assigned to computation and communication / storage devices, respectively. For example, a task mapped on a hardware accelerator is a hardware-implemented function whereas a task mapped over a CPU is a software implemented function.

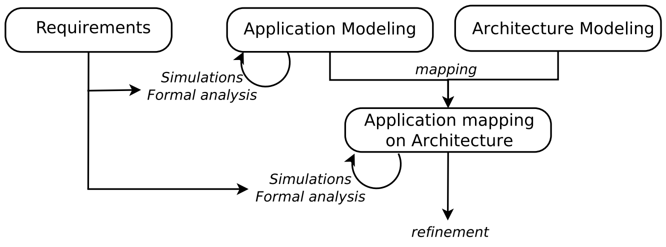


Fig. 1: The Y-chart methodology

The result of this process shall be an optimal hardware / software architecture with regards to criteria at stake for that particular system (e.g., cost, area, power, performance, flexibility, reliability, etc.). There, requirements are to be captured along the process.
This partitioning step is of utmost importance. Indeed, if critical high-level design choices are invalidated afterwards because of late discovery of issues (performance, power, etc.), then it may induce prohibitive re-engineering costs and late market availability.

C. MDE and Embedded Systems

Model-Driven Engineering is probably the main contribution of the last decade in modeling approaches. MDE targets system analysis, design, simulation, code generation, and documentation. MDE generally relies on the UML language, and on meta-modeling techniques in order to define Domain-Specific Languages. OMG's Model Driven Architecture for instance specifically targets two abstraction levels, namely the Platform-Independent Model (PIM) and Platform Specific Model (PSM): embedded systems are thus clearly targeted by MDE. Profiles have also been defined by the OMG to more specifically address embedded systems: SPT [3] and MARTE [4], but none of them addresses requirements modeling. Conversely, the SysML OMG profile [5] clearly takes into account requirements with explicit modeling capabilities and diagrams, but ignores some problematics inherent to embedded systems, e.g., the partitioning issue.

Other methodologies, like for example *Extreme programming* (XP) [6] or *Agile Software Development* [7] have also been proposed to develop software-oriented systems. However, their software focus means that they totally ignore the partitioning issue. They also make traceability and refinement extremely hard to achieve, also because requirements are mostly separate from the design process.

Finally, new software-centric graphical engineering techniques have been introduced during the last decade, and in particular the MDE/MDA approaches standardized at OMG. Our contribution takes security requirements into account. Last but not least, our contributions are settled upon the most recent OMG modeling languages: SysML for the requirement dimension, and MARTE for the partitioning issue.

D. Profiles and Tool Support

To support the modeling of embedded systems, we have already integrated in the same toolkit three MDE-based environment.

- The DIPLODOCUS UML profile [8] specifically targets the partitioning phase. Although usual functional properties are usually studied at application level, performance properties are investigated after mapping, e.g., resource sharing, that is, the scheduling on CPUs (can the architecture execute tasks on time), bus load (can a bus transmit the required amount of data), and properties related to power consumption and silicon area.
- AVATAR [9] relies on an iterative V-cycle for software-based systems, and addresses the analysis, design, implementation and prototyping stages within a SysML environment, thus being used after the partitioning phase. Time interval, non-deterministic choices, synchronizations are enriched with regards to their SysML definitions, making AVATAR particularly suited to model constraints of embedded systems at a high level of abstraction.
- TEPE [10] defines ways to model safety-oriented requirements and properties within a SysML approach.

All those environments are implemented within the free software TTool [11]. TTool automates the formal verification and simulation of models. For DIPLODOCUS, the application or mapping models are taken as inputs, simulation code (C++, SystemC) or formal specifications (LOTOS, UPPAAL) being produces as outputs. TTool also allows to control the simulation in real time (stepwise execution, etc.) and provides live feedback to UML diagrams. Models are simulated with respect to the underlying hardware, as opposed to state-of-the-art UML model simulators which usually operate at a purely functional level. Moreover, the environment does not require a broad knowledge of simulation or formal proofs techniques.

III. Identifying, Analyzing, and Refining Security Requirements

A. Security Goals and Requirements

Haley et al. [12] have shown that the meaning of a security requirement significantly varies between authors. As they point out, security requirements must be precise enough and should make it possible to describe what objects we need to protect and why, and to describe how architectural vulnerabilities are addressed. Haley et al. conclude that one should follow a goal-based approach together with the description of a context which is strongly connected with the execution environment. They distinguish between security goals and requirements: the latter “express the systems security goals in operational terms, precise enough to be given to a designer/architect” and provide “a specification”. Though we roughly agree with this point of view, we do not draw such a clear-cut line. From our point of view, a large part of the security requirements engineering process for embedded systems is to discover at which level in the architecture of the system to introduce a particular security mechanism. We believe that this can only be achieved through the repeated refinement of the architecture, until all relevant security assets are discovered. In an embedded system too, the architecture describes the context and is an essential part of the design.

B. Security Assets

Haley et al. [12] further describe assets as a central notion behind the definition of security goals. In particular, finding an asset leads to envisioning threats to it, and choosing particular security principles to apply to secure it. We agree with this perspective, though we suggest to assess the type of security asset considered, which will affect the nature of related security requirements in an embedded system. In particular, we classify assets into three categories: (1) hardware components (processors, memories, communication channels, clocks, ...), including the data (and secrets) they contain; (2) software assets (virtualization mechanisms, drivers, protocol stacks, applications, ...); (3) information and event flows, which capture communication: those flows reflect the composition of the functions used to realize a particular service together with communication functions to transfer their data and their control events. Such a flow can typically be represented on a DIPLODOCUS UML deployment diagram. Figure 2 illustrates for instance a subset of the functional view - at the partitioning stage - of the flow that would lead a car to automatically brake in case of an emergency in the EVITA system. Relationships between those elements (assets, attacks, and requirements) will be defined by a model as we describe farther below.

C. Security Threats and the Architecture

Security threats define situations that may lead to a system failure, that is a successful attack, in the presence of a malicious behavior. We are interested in describing which assets must be compromised by an attacker in order to perform a successful attack. The identification of a security threat not covered by a the security goals over the targeted security assets clearly points out an incomplete elicitation of security requirements. The designer typically has to either reconsider the mapping of the security goal to assets (as described farther below) or identify missing or incomplete security objectives. On the other hand, while the identification of threats is worthwhile for validating security objectives, we do not put specific requirements that they be exhaustively identified. In particular, security requirements might simply arise from the need to be interoperable with a certain security standard or even out of the certification of the system against a protection profile, as defined for instance in the Common Criteria scheme [13].

Attacks consist of a list of actions over assets (read a message on a bus, etc.) They are very commonly modeled with a wide variety of attack trees. We represent a similar concept in a SysML model that maps attack phases of a given threat to architectural constructs. Figure 3 depicts how this mapping is achieved thanks to a simple extension of the parametric diagram. In particular, each block represents an asset onto which attacks are mapped. Some methodologies aim at defining security threats as a fine-grained refinement of the goals of an attacker, like KAOS’ antigoals [14]. While it is fine to stepwise refine threats and finally attack steps, we consider that this is not enough to capture all architectural mappings. Plain refinement is satisfactory for attacks that only address computational units and communication channels which are seen as blocks in the parametric diagram. However, the information and event flows featured by embedded systems are distributed over the architecture. Attacks on these assets can be modeled as the combination of attacks on the two previous types of assets: for instance, communication between two ECUs might be threatened by an attacker taking control of an intermediate gateway, then dropping all messages arriving on a channel. We therefore introduced a few additional parametric operators to the SysML specification like AND, OR, and AFTER that make it possible to describe the distributed attack scenarios we encounter in embedded systems.

D. SysML for Capturing Security Goals

In SysML, requirements can be captured within SysML Requirements Diagrams. We have specialized the general-purpose SysML requirements by adding a security kind

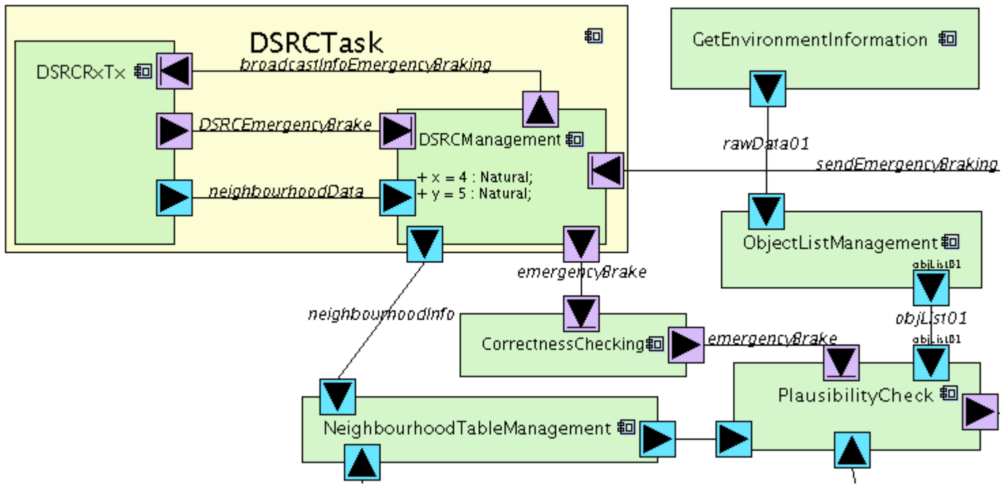


Fig. 2: Information and event flow of the EVITA *emergency braking* use case (UML deployment diagram excerpt)

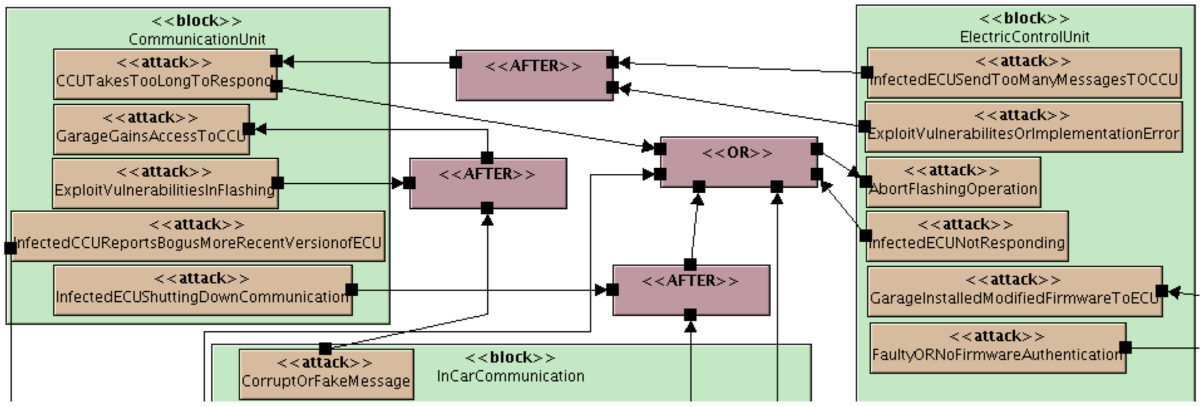


Fig. 3: Attacks mapped to the architecture - EVITA *firmware update* use case (SysML parametric diagram excerpt)

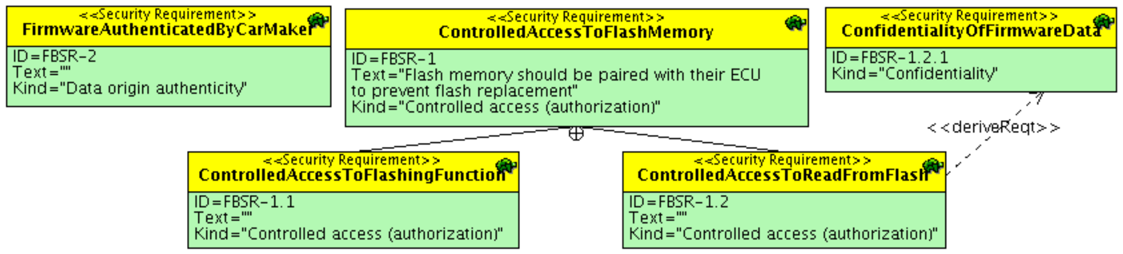
(e.g., privacy, confidentiality, authenticity, integrity, non-repudiation, controlled access, availability, immunity, freshness) to them. The SysML relationships between requirements and defined as follows apply as well to security requirements:

- **Composition**. A requirement is composed of sub-requirements. For example, in Figure 4, *ControlledAccessToFlashMemory* is composed of two sub requirements: *ControlledAccessToFlashingFunction* and *ControlledAccessToReadFromFlash*. Semantically speaking, R_1 is composed of requirement R_2 and R_3 means that R_1 is more abstract than R_2 and R_3 and that the R_2 and R_3 must be satisfied for R_1 to be satisfied.
- **Derive Requirement (DeriveReqt)**. A requirement derives from other requirements, that is, its definition is strongly linked to another one. For example, in Figure 4, *ControlledAccessToReadFromFlash* derives from ConfidentialityofFirmwareData, i.e. the access to flash reading is controlled because the data contained in the flash must be confidential. Semantically speaking, R_1 derives from R_2 means that the security definition provided in R_1 relies on the one which is given in R_2.
- **Copy** is used to re-use the same requirements in several diagrams or views.

E. Mapping Security Requirements to the Architecture

Even with our extension of the SysML requirement construct with additional semantics, requirements still remain unrelated to assets. As explained before, assets in embedded system are composed of functional and architectural elements of the system, which lead us to map requirements to assets at the partitioning phase. This additionally gives us the possibility to relate requirements and attacks, since attacks can only be defined from assets.

The introduction of new assets may obviously lead to the definition of new requirements. For example, introducing flash memories at the partitioning phase may lead to define requirements dedicated to the security of the software code stored in these flash memories (as pointed, for example, in Figure 4). Introducing flash memories however does not mean that all of them are to store confidential software codes. Those two examples demon-



<<Security Requirement>>
FirmwareAuthenticatedByCarMaker
ID=FBSR-2
Text=""
Kind="Data origin authenticity"
<<Security Requirement>>
ControlledAccessToFlashMemory
ID=FBSR-1
Text="Flash memory should be paired with their ECU to prevent flash replacement"
Kind="Controlled access (authorization)"
<<Security Requirement>>
ConfidentialityOfFirmwareData
ID=FBSR-1.2.1
Kind="Confidentiality"
<<deriveReqt>>
<<Security Requirement>>
ControlledAccessToFlashingFunction
ID=FBSR-1.1
Text=""
Kind="Controlled access (authorization)"
<<Security Requirement>>
ControlledAccessToReadFromFlash
ID=FBSR-1.2
Text=""
Kind="Controlled access (authorization)"

Fig. 4: Security requirements in the EVITA *firmware update* use case (SysML Requirement diagram excerpt)

strate two important points: (1) New requirements may be identified when studying candidate architectures at the partitioning phase. (2) Requirements need to be related to architectural elements so as to precise which requirements apply to which architectural elements, and reciprocally.

SysML offers two main facilities to link elements of different natures: the *depend* and *allocate* relationships. The first one models an explicit dependence between a master element and a slave element. The allocate relationship is rather used to mention resources that are provided to an element for its execution, like for instance a task allocated to a processor in the mapping phase. Finally, relationships between requirements and architectural elements are expressed with a dependency link that can be reduced to a simple text field in the requirement box as "referred elements".

F. *Refining Requirements and Threats*

Model-Driven Engineering clearly refers to levels of abstractions, and suggests to rely on model transformation techniques to handle the transition between two different levels. The refinement of elements of the system (e.g., splitting a function into two sub functions, splitting a computing hardware domain into two sub domains) is likely to impact the definition of requirements and attacks. Refining requirements and attacks, or identifying new ones, may also impact the definition of the system design itself. SysML offers no specific facilities for handling refinement processes. In consequence, we have defined our own set of refinement rules [15] to detect elements of the overall model (system design, requirements, attacks) that may be impacted when an element of the architecture (e.g., functions, ECUs, ...) or a security requirement is modified. Those rules are based both on dependencies between elements and on relation semantics (e.g., semantics of the SysML DeriveReqt relationship). However, we do not handle refinements until the point where diagrams can be formally verified: DIPLODOCUS and AVATAR diagrams have a formal semantics, but this is not the case of the description of security requirements that remains informal (like plain SysML requirements).

IV. Evolving the Architecture Based on Security Requirements

A. *From Security Requirements to Security Mechanisms*

The security requirements expressed will determine which mechanisms, like cryptography or access control for instance, will need to be implemented. However, it is likely that these mechanisms cannot be entirely defined until the end of the architecture specification. We argue that it is still possible and desirable to introduce a description of broad security functionalities into the architecture in order to bridge the gap between the requirements analysis phase and the architecture specification phase. In particular, it is important to describe where code for implementing a security-related algorithm will be run, and how resilient its execution will be. The hypotheses made about implementation resilience when defining requirements should be transferred to the hardware/software partitioning model. By doing so, we also associate a security requirement to an architectural asset, and thus precise which threat may its realization may be exposed to: for instance, implementing the access control to the Head Unit of the car with a hardware based implementation will, for remote attackers, be enough to mitigate all attempts at bypassing this control. We consider this mapping process as essential to achieving a more complete and accurate threat and risk assessment analyses, and to eliciting other security requirements. Finally, this mapping may even reveal the impossibility of any satisfactory implementation.

Finding which assets are at risk also contributes to the definition of the defense perimeter achieved by a related security objective, and hence to the placement of the security mechanism that derives from this objective. Typically the possibility of injecting fake traffic from the driver's mobile phone onto the vehicle backbone bus might suggest to filter out mobile phone messages in the engine or chassis and safety domains. This might in turn be implemented through either the authentication of the sender using cryptography or through access control at respective domain gateways based on the origin of traffic.

Due to the communication-centric nature of the threats that we focus on, the introduction of security functionalities will also modify the purely functional information and event flows of the embedded system. We introduce security through the addition of security functions to existing entities, or through the introduction of new entities. Those new entities would typically feature specific security properties that should be matched by the final implementation in the architecture: for instance one such additional entity might reflect the need to use of a tamper-resistant hardware module (and subsequent communication with it). Those additional security mappings to the architecture might be introduced within sequence diagrams, or better, in deployment diagrams.

B. Detecting Conflicting Classes of Requirements

Security requirements may typically impact other classes of requirements, especially non-functional ones like the interoperability, or more critically for an embedded system, safety. This may result in conflicts. For instance, an excessively large security payload may reduce the available bandwidth of a bus and endanger the safety critical transmission of an emergency braking notification between the brake sensor and the brake actuator. Similarly, security may incur additional costs, like for the verification of the signature of a message, that may entail an an overly long latency for a low-cost ECU. Such conflicts have to be detected.

We consider that, because it focuses on security assets which are architecture-centric, our methodology is compatible with other requirements engineering methodologies in which the requirement can be linked to the architecture. While we did not experiment with the integration of existing engineering methodologies for other types of requirements, we validated the satisfaction of a few safety requirements through the generation of tests and simulations from our embedded system model. For instance, we simulated the CAN bus to evaluate the impact of a cryptographic envelope implementing our authentication requirements. We also evaluated the braking latency in the EVITA emergency braking use case by automatically generating software for a virtual prototyping environment from AVATAR models. Some of these validations can be described even at the goal level description of security requirements through the use of SysML observers.

V. Methodology: Summary and Evaluation

A. Summary

The overall methodology we are proposing captures the design objectives described in the previous sections flexibly with the evolution of the architecture specification. We can summarize its different phases with the following iterative process:

1) **Initial Architecture Mapping.** One or several typical use cases are selected as a starting point for investigation. The functionalities of the system highlighted in these use cases are first modeled as tasks. Exchanges between functions are modeled with information and event flows. Event-based communications is also captured in order to control the Information Flow. Tasks and communication can then be mapped to a draft architecture of the system, following the Y-chart approach. Designer experience plays a key role for determining first draft architectures. Finally, assets are selected from this architecture.
2) **Security Objective Identification.** Security objectives might be identified (1) from the above use cases, for instance because of imposed standards or of the properties expected from the system, or (2) from unaddressed threats or attacks on assets, or (3) as the refinement of another security objective when the process is iterated and the level of detail of the architecture has changed. They are represented using the SysML requirements diagram. In further iterations, one may need to delete or adapt security objectives deprecated by modifications on the architecture.
3) **System Asset Identification.** System assets are identified among architectural elements (processors, pieces of software, communication channel). Assets can specifically refer to a particular element, yet we recommend the use of generic assets, like for example: “all system busses”.
4) **Threat Identification.** Threats and security vulnerabilities of the selected assets are identified using our extension of the SysML parametric diagram. Threats should as much as possible document the capabilities that an attacker should meet or exceed, as well as information about the origin of attacks (local, remote, through a specific interface).
5) **Risk assessment.** This step requires a strong security expertise. Techniques for performing this analysis are outside the scope of this paper, but have been largely described in the literature. We also applied a specific metric to evaluate requirements in the EVITA case study in [16]. Although we haven’t automated this step, we have plans to model risk metrics by introducing specific attributes in SysML parametric diagrams. Risk assessment typically relies on the definition of an attacker model.
6) **Threat Coverage and Prioritization.** One should verify the coverage of threats or attacks by security objectives (TTool automates that verification). Based on the risk analysis, one should also categorize and prioritize security objectives that are mapped to a threat . The most important security requirements will be further refined, while other requirements may be left aside or even abandoned at this stage. Those objectives are kept in SysML diagrams but duly annotated. Consistency checks should also be performed to ensure that a security objective does not conflict with another security requirement expressed over the same asset.
7) **Architecture Refinement.** The architecture, including the specification of assets, is refined. This refinement will result on the one hand from a more

detailed description of the architecture components as use cases or the architecture are becoming more precise (e.g., new communication channels, refinement of an execution environment into OS/middleware/application layers, etc.). On the other hand, it may also result from linking requirements to system information flows. Finally, the ongoing process is iterated to the identification of new security objectives.

B. *Evaluation*

The EVITA case study we presented in the beginning of this paper was modeled using the TTool open-source toolbox. Result of this study can be found on the EVITA project web site [2] and browsed with the toolbox. Details about this study and the related risk analysis can also be found in [16]. EVITA includes around 20 case studies. Out of the mass of use case specific requirements obtained by applying our methodology, we could extract around 32 general requirements, i.e., requirements that apply to all case studies. These requirements have been split into four categories: availability (7 requirements), privacy (7 requirements), fake commands (9), and environment-related requirements (9). General requirements have been progressively extracted from requirements specific to the first analyzed case studies. We have then verified their relevance for other case studies. They could probably constitute a good foundation for defining security-oriented patterns. Then, for each case studies, we have defined specific requirements. For example, for the *firmware update* use case, 9 additional security requirements have been elicited: some of them are displayed in Figure 4. Both general and specific requirements are linked to assets and attacks. The iteration between requirements, architecture, and attacks also led us to identify much more attacks than when simply identifying them based on our security expertise and on brainstorming sessions: the number of identified attacks was multiplied by a factor between 2 and 4, depending on use cases.

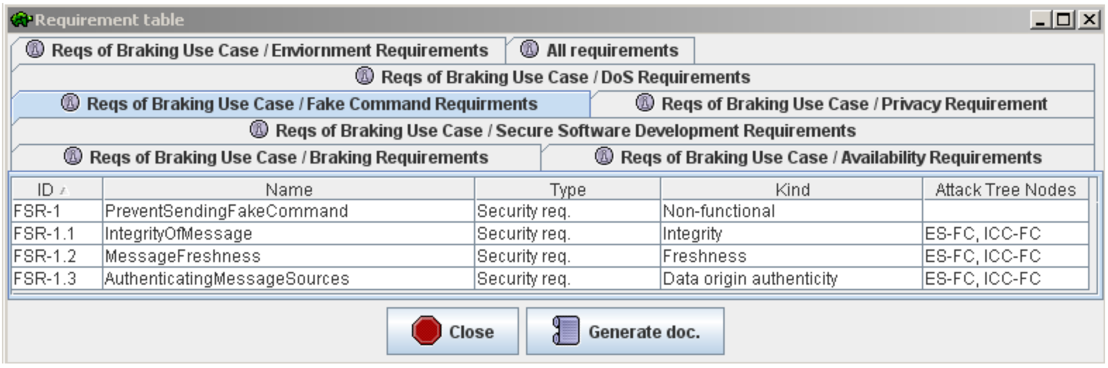
Lessons learned. One of our initial objectives was to achieve a comprehensive and more systematic analysis of security issues of an automotive on-board system. Based on our experience with the EVITA case study analysis, we can confidently state that our methodology has significantly improved the elicitation of security requirements. The adherence to a stepwise and iterative process helped us a lot to resist the constant urge to jump directly to the selection of particular security mechanisms to map to our system. This would have likely resulted in a less complete insight into the threats to the vehicle, and in a poorer understanding of the dependences between security requirements. Similarly, the use of the architecture and function placement helped us not only to understand the use cases, but also to point at some inconsistencies regarding security requirements they expressed.

Our methodology aims at guiding and providing representations of security requirements compatible with the MDE approach of practitioners that are not security specialists. This has been the reason for our adoption of the OMG standards, which are quite widespread in the embedded system world today. In contrast, the result of our analysis can be compared with the work of Bar-El [17] on automotive on-board networks. This work in particular essentially focuses on cryptographic mechanisms supporting the architecture. We contend that our analysis has better captured security requirements and their dependencies than the text analysis in this work. Furthermore, this work fails to address the hardware/software partitioning issue or the compatibility of security mechanisms with safety requirements, that are so important in vehicles. In comparison, we relied on the mapping of security requirements with assets and with the information flows to later evaluate the satisfaction of safety, realtime, or performance requirements through formal validation [18], as well as through tests and simulations in the same open-source toolbox [19].

The definition of requirements in our case study also convinced us early on that scalability had to be addressed. To this end, we are using SysML *copy* relationships between different SysML diagrams, which reference other requirements. This approach makes it possible to organize requirements with use-case oriented views. We detected during our definition the occurrence of security design patterns that could be systematically applied to elicit finer-grained requirements. For instance, whenever we were referring to ensuring the secure communication of two entities, we could always derive an authentication, an integrity, and a freshness requirements over some information or event flow. We also found that a similar strategy applied to the elicitation of attacks. Scalability also relates to the automation of consistency checks. In particular, we implemented in the toolbox an automated verification of the coverage of attacks with security objectives, as depicted in Figure 5.

Limitations. Our methodology does not bridge the implementation gap. In particular, we do not describe requirements as to the effectiveness of the implementation of the security mechanisms that realize the requirements described. Most notably, secure programming issues, like buffer overflows, are plaguing all kinds of software produced. Yet they are not the result of a poor design of the architecture as we can describe it in SysML and UML. Similarly, hardware based implementations can be poorly done, and for instance be vulnerable to timing or power attacks. We believe that addressing such issues is not an architecture design issue per se, and can be achieved by the systematic application of software or hardware engineering recipes. Nor our methodology nor the tools we developed

[2]The corresponding files are available at the following link: http://www.evita-project.org/Deliverables/evita_t2300_23.xml


Requirement table
Reqs of Braking Use Case / Enviornment Requirements
All requirements
Reqs of Braking Use Case / DoS Requirements
Reqs of Braking Use Case / Fake Command Requirments
Reqs of Braking Use Case / Privacy Requirement
Reqs of Braking Use Case / Secure Software Development Requirements
Reqs of Braking Use Case / Braking Requirements
Reqs of Braking Use Case / Availability Requirements
Close
Generate doc.

ID	Name	Type	Kind	Attack Tree Nodes
FSR-1	PreventSendingFakeCommand	Security req.	Non-functional	
FSR-1.1	IntegrityOfMessage	Security req.	Integrity	ES-FC, ICC-FC
FSR-1.2	MessageFreshness	Security req.	Freshness	ES-FC, ICC-FC
FSR-1.3	AuthenticatingMessageSources	Security req.	Data origin authenticity	ES-FC, ICC-FC

Fig. 5: Security Requirements Coverage Table (automated consistency check in TTool)

can provide support for these tasks though.

The approach we have proposed is also not addressing all aspects of the automation of security requirements engineering in embedded systems. In particular, SysML still lacks the way to describe a terminology of requirements and relationships that would make it feasible to designate unambiguously and to reason over the structure of requirements and to automate the consistency checking process. Although these techniques are outside the scope of this paper, we have integrated security ontologies into TTool as a means to achieve both objectives.

VI. Related Work

There has been a quite remarkable progress in the area of security requirements engineering in the past decade. In [20], Nhlabatsi et al. classify security requirements engineering work in software systems according to four dimensions, namely: (1) *goal-based approaches*: the KAOS framework [14] was the first such approach to model, specify, and analyze security requirements as the organized refinement of goals into a set of sub-goals using generic patterns. (2) *Model-based approaches*: in contrast, those methodologies, like UMLSec [21], focus on the mapping of security mechanisms to the software architecture. (3) *problem-oriented approaches* focus on the expression of attacks and on the threat-based elicitation of security requirements. They are represented by approaches like abuse frames [22] or misuse cases [23]. (4) *Process-oriented approaches*, like the SQUARE [24] methodology, finally aim mainly at the risk analysis of an existing design and follow a rather rigid waterfall approach to engineering, yet do not address well design exploration and refinement.

Those approaches are not incompatible, and our own proposal essentially associates a goal-oriented description of security objectives with a model-driven approach to system design and in particular to the refinement of security assets. In particular, we benefit from the refinement and tracing qualities of goal-based approaches that have been similarly successful in other domains of requirement engineering. Another strength of goal oriented approaches lies in their ability to capture dependencies even between security requirements with a high-level of abstraction. Models on the other hand are extremely good at capturing architecture details and have been shown an excellent fit for describing security requirements regarding cryptographic protocols. For instance, security requirements and functions can be progressively refined until a formal verification step integrated in our toolkit [9]. TTool implements model transformation techniques in order to translate refined and formally expressed security requirements and designs into a pi-calculus specification taken as input by ProVerif, a Dolev-Yao based security proof toolkit [25].

It is worth mentioning that the model-driven engineering of requirements has long been supported by researchers in the field of embedded systems [26], [27], [28]. However, only Peraldi et al. [29] advocate the need to link the model driven engineering of the system architecture and a goal-oriented expression of requirements that we follow in our approach. To our knowledge, none of these proposals has addressed the expression of security requirements.

Our methodological proposals also share quite some similarities with the TwinPeaks approach advocated by Nuseibeh [30], although the latter only deals with software systems. Instead of a simple spiral alternating between the requirements and the architecture as TwinPeaks suggests, we alternate between the Y-Chart modelling of software and its mapping to hardware components, the identification of assets and threats to them, and the identification of security requirements. In particular, we also deal with the three management concerns that TwinPeaks aims at addressing: (1) exploring the solution space (in our case, both the embedded system architecture and attacks that may result out of this design) early makes it possible to incrementally provide feedback about requirements; (2) the designer has to rely on commercial off-the-shelf software (as for TwinPeaks), or available electronic components, or standard cryptographic algorithms and requirements (security requirements in our proposal) help narrow down their proper selection; (3) rapid change, which is also very much linked with refining the architecture in our case.

Finally, Haley et al. [12] suggest to use propositional logic to reason about the satisfaction of security requirements and their context. We believe this approach is quite interesting and plan to develop a logical framework for reasoning about requirements in our own system. Regarding the validation of requirements, we have also investigated the assessment of the impact on safety of the security mechanisms introduced after security requirements, like for instance, assessing the added latency for performing a braking operation with secure communication. There as well, the use of an architecture-centric model of the system makes it possible to proceed to tests and simulations.

VII. Conclusion

We have introduced both a methodology to define security requirements in an embedded system and an open-source tool to support this process. We conducted an experiment on an automotive on-board system under definition, which helped us assess the adequacy of our approach. Out of this experience, security requirements should be influenced by the system's architecture and in turn, influence its structure through a refinement process integrated to the embedded system lifecycle, through the Y-chart approach in our case. Our main claim is that a security requirements engineering methodology should capture this relationship. Assets are central to defining security requirements and we argue that they are even more critical in an embedded system where they encompass many elements: they are to be found in the system architecture components (both hardware and software), in their particular mapping, and in communications within the system and through external interfaces. We believe that our adoption of a model-driven refinement of security assets might be used right from the early definition of security requirements to link the security expert's goal-oriented point of view with the model-centric perspective of the embedded system designer and enable their collaboration before any of them commits to an inappropriate solution.

We plan to further investigate security design patterns that would be specific to embedded systems in order to ensure a more systematic derivation of security requirements. Explicitly representing such patterns with a particular construct in our models might be a way to produce more compact diagrams. We are also working towards a more comprehensive and automated application of our methodology, encompassing different classes of requirements (security, safety, performance, etc.). In particular, we plan to formalize the meta-model of the relationships between those different requirements so as to introduce security-oriented reasoning capabilities in our modeling environment.

References

[1] E. Kelling, M. Friedewald, T. Leimbach, M. Menzel, P. Säger, H. Seudié, and B. Weyl, "Specification and Evaluation of e-Security Relevant Use cases," EVITA Project, Tech. Rep. Deliverable D2.1, 2009.

[2] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, "Metropolis: an integrated electronic system design environment," *Computer*, vol. 36, no. 4, pp. 45–52, April 2003.

[3] OMG, "Omg profile for scheduling, performance and time," in *http://www.omg.org/spec/SPTP/*, 2005.

[4] J. Vidal, F. de Lamotte, G. Gogniat, P. Soulard, and J.-P. Diguet, "A co-design approach for embedded system modeling and code generation with UML and MARTE," in *Design, Automation and Test in Europe Conference and Exhibition, 2009. DATE'09*, April 2009, pp. 226–231.

[5] OMG, "Omg systems modeling language," in *http://www.sysml.org/specs/*, 2012.

[6] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[7] K. Waters, *All About Agile: Agile Management Made Easy!* CreateSpace Independent Publishing Platform, 2012.

[8] L. Apvrille, W. Muhammad, R. Ameur-Boulifa, S. Coudert, and R. Pacalet, "A UML-based environment for system design space exploration," in *Electronics, Circuits and Systems, 2006. ICECS '06. 13th IEEE International Conference on*, Dec. 2006, pp. 1272 –1275.

[9] G. Pedroza, D. Knorreck, and L. Apvrille, "AVATAR: A SysML environment for the formal verification of safety and security properties," in *The 11th IEEE Conference on Distributed Systems and New Technologies (NOTERE'2011)*, Paris, France, May 2011.

[10] D. Knorreck, L. Apvrille, and P. De Saqui-Sannes, "TEPE: A SysML language for time-constrained property modeling and formal verification," *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 1, pp. 1–8, Jan. 2011.

[11] L. Apvrille, "TTool website," in *http://ttool.telecom-paristech.fr/*, 2013.

[12] C. Haley, R. Laney, J. Moffett, and B. Nuseibeh, "Security requirements engineering: A framework for representation and analysis," *Software Engineering, IEEE Transactions on*, vol. 34, no. 1, pp. 133 –153, jan.-feb. 2008.

[13] T. C. C. R. Agreement, "Common criteria for information technology security evaluation part 1 : Introduction and general model july 2009 revision 3 final foreword," *NIST*, vol. 49, no. July, p. 93, 2009.

[14] A. Van Lamsweerde, "Engineering Requirements for System Reliability and Security," *Software System Reliability and Security*, vol. 9, pp. 196–238, 2007.

[15] M. S. Idrees, "A requirements engineering driven approach to security architecture design for distributed embedded systems," Ph.D. dissertation, Telecom ParisTech, 2012.

[16] A. Ruddle and et al, "Security Requirements for Automotive On-board Networks Based on Dark-side Scenarios," EVITA Project, Tech. Rep. Deliverable D2.3, 2009.

[17] H. Bar-El, "Intra-vehicle information security framework," *Design*, pp. 1–19, 2009.

[18] G. Pedroza, M. S. Idrees, L. Apvrille, and Y. Roudier, "A formal methodology applied to secure over-the-air automotive applications," in *VTC-Fall2011, IEEE 74th Vehicular Technology Conference, 5-8 September 2011, San Francisco, USA*, San Francisco, ÉTATS-UNIS, 09 2011. [Online]. Available: http://www.eurecom.fr/publication/3484

[19] B. Weyl, M. Wolf, F. Zweers, T. Gendrullis, M. S. Idrees, Y. Roudier, H. Schweppe, H. Platzdasch, R. E. Khayari, O. Henniger, D. Scheuermann, A. Fuchsa, L. Apvrille, G. Pedroza, H. Seudie, J. Shokrollahi, and A. Keil, "Secure On-board Architecture Specification," EVITA Project, Tech. Rep. Deliverable D3.2, 2010.

[20] A. Nhlabatsi, B. Nuseibeh, and Y. Yu, "Security requirements engineering for evolving software systems: a survey," The Open University, Tech. Rep. 1, 2010. [Online]. Available: http://www.igi-global.com/Bookstore/Article.aspx?TitleId=39009

[21] J. Jürjens, "UMLsec: Extending UML for Secure Systems Development," *5th International Conference on the Unified Modeling Language*, pp. 412–425, 2002.

[22] L. Lin, B. Nuseibeh, D. Ince, M. Jackson, and J. Moffett, "Introducing abuse frames for analysing security requirements,"

in *Proceedings of the 11th IEEE International Conference on Requirements Engineering*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 371–. [Online]. Available: http://dl.acm.org/citation.cfm?id=942807.943895

[23] G. Sindre and A. Opdahl, "Eliciting security requirements by misuse cases," in *Technology of Object-Oriented Languages and Systems, 2000. TOOLS-Pacific 2000. Proceedings. 37th International Conference on*, 2000, pp. 120 –131.

[24] N. R. Mead and T. Stehney, "Security quality requirements engineering (square) methodology," *SIGSOFT Softw. Eng. Notes*, vol. 30, pp. 1–7, May 2005. [Online]. Available: http://doi.acm.org/10.1145/1082983.1083214

[25] B. Blanchet, "Automatic verification of correspondences for security protocols," *Journal of Computer Security*, vol. 17, no. 4, pp. 363–434, Jul. 2009.

[26] M. Broy, "Requirements engineering for embedded systems," 1997.

[27] M. von der Beeck, P. Braun, M. Rappl, and C. Schröder, "Model based requirements engineering for embedded software," *2012 20th IEEE International Requirements Engineering Conference (RE)*, vol. 0, p. 92, 2002.

[28] E. Geisberger, J. Grunbauer, and B. Schatz, "Interdisciplinary requirements analysis using the model-based rm tool autoraid," *Automotive Requirements Engineering, International Workshop*, vol. 0, p. 1, 2006.

[29] M.-A. Peraldi-Frati and A. ALBINET, "Requirement traceability in safety critical systems," in *EDCC2010 - Workshop on Critical Automotive applications: Robustness and Safety (CARS'2010)*, ser. ACM International Conference Proceeding Series, J.-C. Fabre, O. Guetta, and M. Trapp, Eds. Valencia, Espagne: ACM, Apr. 2010, pp. 11–14. [Online]. Available: http://hal.inria.fr/hal-00687550

[30] B. Nuseibeh, "Weaving together requirements and architectures," *IEEE Computer*, vol. 34, no. 3, pp. 115–117, 2001.