Impact of Security Measures on Performance Aspects in SysML Models

Maysam Zoor, Ludovic Apvrille and Renaud Pacalet
LTCI, Télécom Paris, Institut Polytechnique de Paris, France
{firstName.lastName}@telecom-paris.fr

Keywords: Embedded Systems, Security, Performance, MBSE, Simulation, Verification.

Abstract: Because embedded systems are now frequently connected, security must be taken into consideration during system modeling. However adding security features can degrade performance. In this paper, the trade-off between security and performance is tackled with a new model-based method that can automatically assess the impact of security measures on performance. The contribution is illustrated with an industrial motor control taken from the H2020 AQUAS project.

1 INTRODUCTION

Ensuring the safety of embedded systems implies to also take into account security threats (Chai et al., 2019) (Jiang et al., 2013). However security measures require extra processing time which might degrade system performance (Chai et al., 2019)(Li, 2018)(Kocher et al., 2004)(Apvrille and Li, 2019).

(Davis, 2014) defines the time constraints as "end-to-end deadlines on the elapsed time between a stimuli and the corresponding response". (Li, 2018) added to that the percentage of usage of system's components to characterize system performance.

Studying security and performance in an isolated manner is not as efficient as considering the dependency between them (Gruber et al., 2018). Also, as stated by (Viehl et al., 2006), estimating performance and evaluating an architectures at an early design stage is considered as "very valuable approach in the area of SoC design".

Our contribution proposes to tackle the interdependency and trade-offs between performance and security requirements of embedded systems at an early design stage. Our new performance analysis approach is based on simulation traces and SysML model analysis. It is expected to allow engineers to adjust better the design in early stages of the product life cycle, leading to a balanced solution with a decrease in the development time and efforts and an increase in product quality (Friedenthal et al., 2014). The novelty of the presented performance analysis method is its ability to provide answers to questions such as (i) which security HW/SW might delay a critical event, (ii) how to map security algorithms (e.g. HW/SW components) and which more general hardware platform should be used, including buses able not to delay critical messages, including secured messages.

These questions form a subset of larger set of questions faced by a designer during a system-level design (Thiele et al., 2007). Answers to these questions will help designers estimate the relation between added security and performance early in the design.

The next section studies related work. Then, section 3 presents the SysML-Sec modeling and verification approach upon which our new contribution is based. Our performance analysis approach is explained in section 4, and illustrated in section 5 with a motor drive system. Finally, section 6 concludes the paper.

2 RELATED WORK

Although the trade off between security and performance in embedded systems has been highlighted since years (Kocher et al., 2004), it wasn't until recently that the impact of added security on system performance started to be studied in the design stage.

Fujdiak et al (Fujdiak et al., 2018) proposes experimental measurements that demonstrate a linear relation between security levels and performance.

(Li et al., 2017) relies on a modeling and verification tool named TTool (Apvrille, 2013) (section 3) to study the security-performance dependency in a disaster relief drone model. In their work, Li et al, compared the total execution cycles of secure and non secure designs to show the performance impacts of security in concrete systems. Yet, designers have

no way to figure out which HW/SW elements really caused extra delays.

To enhance this security-performance trade-off study, the *latency* concept was added to TTool (Li, 2018). TTool can thus now measure the min, max and average latency between (safety critical) events. Yet, the explanation of why the latency has increased when updating a model still relies on manual analysis of — sometimes very long — simulation traces.

Another approach to study the trade-off between security and performance is based on the interaction between two different tools as presented in (Fujdiak et al., 2019). In this interaction, TTool was used to help decide which security level shall be selected with regards to the desired performance by comparing the final system response time of a model for different security levels.

Time analysis can be performed using multiple approaches. These approaches can be classified into different categories. Simulation and formal approaches are the most used ones in the domain of performance estimation of embedded systems (Thiele et al., 2007) (Viehl et al., 2006). While simulation tools and industrial frameworks e.g. Koski (Kangas et al., 2006) can only consider a limited set of execution traces and corner cases are usually unknown (Thiele et al., 2007), formal approaches like timed automata are usually limited on scope to the model under analysis where sharing of resources leading to complex interactions among components is difficult to take into consideration.

To overcome the limitations encountered when using either methods, (Thiele et al., 2007) and (Viehl et al., 2006) combined simulation and formal approaches to analyze system performance. However in these approaches, verifying the security of a model and the impact of the added security method on the functional and architecture aspects is not supported.

The model-based environment TimeSquare (DeAntoni and Mallet, 2012), supporting the Clock Constraint Specification Language (CCSL), provides facilities for performance analysis, in particular for the analysis of execution traces. Yet, it lacks security verification and architecture exploration techniques.

Based on this overview, we will take advantage of the already developed security-performance trade-off analysis within TTool and more precisely its modeling profile named SysML-Sec to study the effect of security features (added computations, added communications and increased message sizes) onto system performance.

3 SysML-Sec

3.1 Method

Sysml-Sec is one of the modeling profiles supported by the free and open-source toolkit named TTool (Apvrille, 2013). Sysml-Sec (Apvrille and Roudier, 2013) extends SysML for the design of safe and secure embedded systems while taking performance aspects into account. SysML-Sec methodology allows system designers to iterate over safety and security features and continuously check if the system requirements still hold. This verification can be done along the three stages of the SysML-Sec method (see Figure 1): analysis, HW/SW Partitioning and Software Design (Apvrille and Li, 2019).

After the analysis stage where requirements attacks and faults are captured, system design includes system-level HW/SW partitioning and software design. First, in the HW/SW partitioning stage, the architecture and high-level functional behavior are modeled at a high-level of abstraction and then linked in the mapping stage, as defined in the Y-Chart approach (Kienhuis et al., 2001). Second, the design of software components is performed (Li, 2018).

Verification can be performed with a press-button approach from most diagrams to determine if the defined requirements are satisfied. TTool can perform verifications using formal techniques (e.g., model-checking) and simulations. Safety verification relies on the TTool model checker. Security verification relies on an external toolkit called ProVerif (Blanchet et al., 2018). Performance verification relies on a System-C like simulator of TTool.

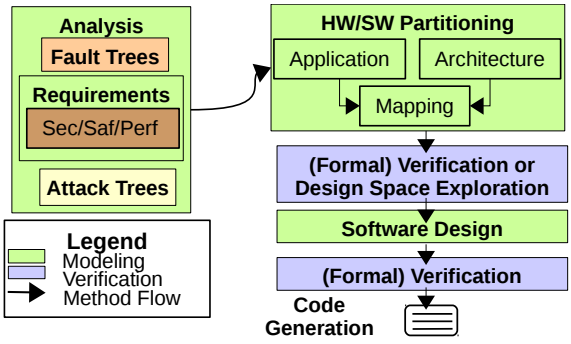

Figure 1: SysML-Sec modeling profile used in TTool.

3.2 HW/SW Partitioning

The paper now focuses on HW/SW partitioning. A HW/SW partitioning P is formally defined as the composition of a Functional Model FM, an Architec-

ture Model AM and a Mapping Model MM (Li, 2018):

$$P = (FM, AM, MM) \quad (1)$$

3.2.1 Application/Functional Modeling

A Functional Model FM is captured with SysML block diagrams. FM has a set of Tasks T and of communications $Comm$ between tasks (Li, 2018):

$$FM = (T, Comm) \quad (2)$$

Each task $t \in T$ has a set of attributes $Attr$ and a behavior $\mathscr{O}$ described with an activity diagram built upon a set of operators:

$$t = (Attr, \mathscr{O}) \quad (3)$$

Operators (equation 4) can be divided into 3 operator categories. Control operators $CtrlOp$ handle the execution flow of a task e.g. loops and tests. Communication operators $CommOp$ refer to Channels, Events or Requests: channels model data exchange, events are used for synchronization and requests model task spawning. Last, Complexity operators $CompOp$ capture computation complexity e.g. in terms of int operations ($ExecI$). An operator $\theta \in \mathscr{O}$ is defined as follows:

$$\begin{aligned}\theta \in \{&StartState, StopState, ForLoop, Choice,\\ &Sequence, SelectEvt, SendEvent, ReadChannel,\\ &WriteChannel, SendRequest, ReadRequestArg,\\ &WaitEvent, ExecI, ExecC, Delay, Random\}\end{aligned} \quad (4)$$

The next operator(s) for each θ are known as a set of $sub_{\mathscr{O}}$. CtrlOp operators may have several next operators depending on the guard conditions. The connection between θ and its next operators in $sub_{\mathscr{O}}$ will be denoted as $ControlFlow_{(\theta \rightarrow sub_{\mathscr{O}})}$.

3.2.2 Architecture Modeling

An Architecture Model AM is built upon a set of hardware nodes and links $Links$ between nodes described with a UML Deployment Diagram. Hardware nodes are split into of three sets: execution ($ExecNode$), communication ($CommNode$) and storage nodes ($StNode$). An Architecture Model is defined as folllows:

$$AM = (ExecNode, CommNode, StNode, Links) \quad (5)$$

3.2.3 Mapping Modeling

A mapping model specifies where tasks and there communications are allocated to hardware components using UML deployment diagrams. Tasks mapped to processors are software implemented while tasks mapped to Hardware Accelerators are hardware implemented. A task t mapped in a partitioning p is denoted as t_p.

4 PERFORMANCE EVALUATION

Simulation of mapping models helps understanding system performance especially after security measures have been added. Yet, understanding the precise cause of extra latencies when adding extra mechanisms requires to improve the ways simulation traces can be investigated. To so do, we introduce a new way (shown in Figure 2) to compare parts of simulation traces (called "simulation trace analysis" or STA) related to an extra mechanism.

1. A simulation trace is computed using TTool's simulator
2. Among the operators $\mathscr{O}$ of all tasks in T, the designer selects two of them between which the latency will be analyzed.
3. The mapping model is transformed into a directed graph so as to analyze the dependencies inside of the simulation trace
4. The Simulation trace, the two operators, and the directed graph are injected into our *simulation trace analyzer* (detailed below).

Of course, all other verification techniques (e.g. verifying security properties) are still available and used for their respective purposes.

4.1 Latency Graph Generation

Latency graphs are directed graphs. A mapping model is transformed into a latency graph G consisting of a set of vertices's v and a set of directed edges ε.

$$G = (v, \varepsilon) \quad (6)$$

A vertex in the set v represents either a task, or a task operator in the functional model, or a hardware node in the architecture model

$$v \subseteq N_{AM} \cup N_{FM} \quad (7)$$

where $N_{AM} \in \{CommNode, StoreNode, ExecNode\}$ and $N_{FM} \in \{T, \theta\}$ An edge in the set ε represents an element of an activity diagram i.e.

$$\varepsilon \subseteq Links \cup Comm \cup ControlFlow_{(\theta \rightarrow sub_{\mathscr{O}})} \quad (8)$$

Algorithm 1 (simplified version: loops and sequences are not shown) builds the graph according to operators listed in equation 4.

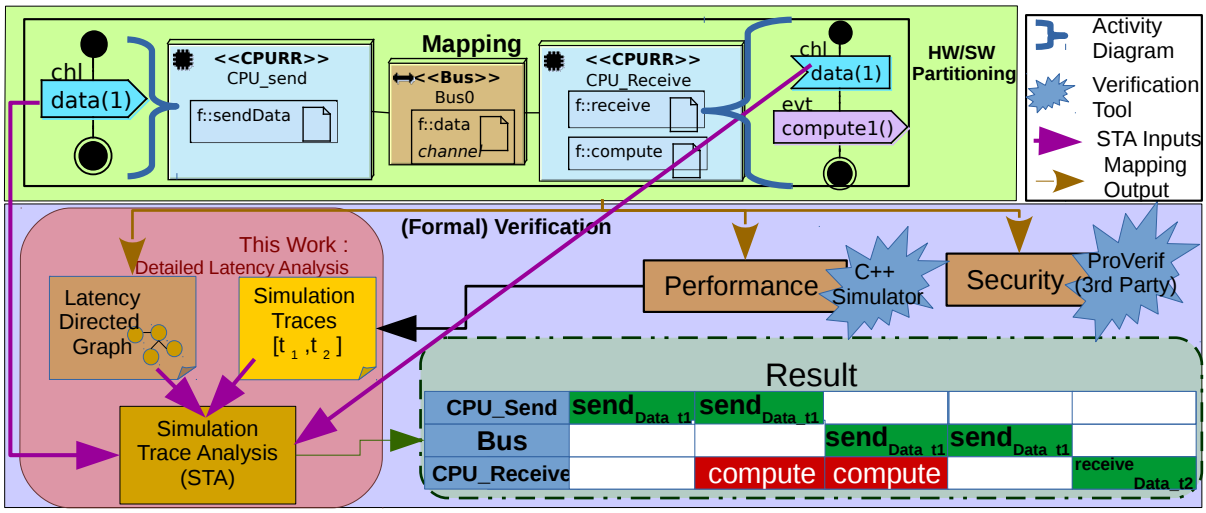

Figure 2: Detailed Latency Analysis Method added to SysML-sec.

4.2 Detailed Latency Analysis Method

4.2.1 Simulation Transaction

TTool's simulator for mapped models is transaction-based (Knorreck, 2011), and takes into account both HW and SW elements. For short, a simulation transaction contains the following attributes:

- **device**, **task** and **operator**: *device* executed an *operator* of *task*
- **startTime** and **endTime** (in clock cycles)
- **length**: number of clock cycles needed to execute the transaction

4.2.2 Simulation Trace

The simulation of a mapped model p for a time interval $[t_1, t_2]$ outputs a simulation trace $s_p^{t_1,t_2}$ built upon n simulation transactions st_x:

$$s_p^{t_1,t_2} = \{st_1, \ldots, st_n\} \tag{9}$$

4.2.3 Latency between Operators

Let θ_A andθ_B be the two operators between which the latency is studied in s_p. All transactions $st \in s_p^{t_1,t_2}$ where $st.operator = \theta_A$ are added to Occ_{θ_A} (ordered by start time). Same for Occ_{θ_B}. Equation 10 defines Occ_{θ_A}.

$$Occ_{\theta_A} = \{st_{0_A}, st_{1_A}, \ldots, st_{n_A}\} \tag{10}$$

The time delay between the i^{th} transaction in Occ_{θ_A} and Occ_{θ_B} referred to as latency λ_i can be calculated as:

$$\lambda_i = endTime_{(st_{i_B} \in Occ_{\theta_B})} - startTime_{(st_{i_A} \in Occ_{\theta_A})} \tag{11}$$

To compute λ_i, Occ_{θ_A} and Occ_{θ_B} should each contain a transaction at the i^{th} position related to θ_A and θ_B respectively. In case this condition is not satisfied, or in case the designer is interested in the max and min latencies, the algorithm for calculating min and max latencies (Li, 2018) is applied on Occ_{θ_A} and Occ_{θ_B}. TTool can now display λ_i as well as λ_{min} and λ_{max} representing the minimum and maximum latency values between Occ_{θ_A} and Occ_{θ_B}.

4.2.4 Latency Analysis

Algorithm 2 can compute the reason for the time delay between the i^{th} occurrence of the selected operators. It first extracts only all relevant transactions i.e. the ones related to the studied latency. To do so, the algorithm searches the trace s_p for transactions that occurred between the $startTime_{st_{i_A}}$ and $endTime_{st_{i_B}}$ denoted as s'_p. It takes as input θ_A, θ_B, $startTime_{st_{i_A}}$, $endTime_{st_{i_B}}$, s_p, and the generated latency graph G. When the considered time delay is λ_{min} or λ_{max}, the same algorithm (algorithm 2) is used however $startTime_{st_{i_A}}$ and $endTime_{st_{i_B}}$ are replaced by either $startTime_{st_{min_A}}$ and $endTime_{st_{min_B}}$ or $startTime_{st_{max_A}}$ and $endTime_{st_{max_B}}$ respectively. However not all transactions in s'_p are directly related to the delay between $startTime_{st_{i_A}}$ and $endTime_{st_{i_B}}$. In order to extract which transactions are involved in this delay, we rely on graph g: we first build all possible paths Pa from θ_A to θ_B. For each transaction in $st \in s'_p$, if st belongs to Pa then we consider that st is mandatory i.e. it contributes directly to the delay λ_i, thus it is added to an array named $onPathTransactions$. In case there is no path between $node_{\theta_A}$ and $node_{\theta_B}$, then a transaction is considered

Algorithm 1: Latency Graph Generation.

```
   Data: Mapping Diagram
   Result: Latency Graph
 1 foreach node in AM | node
     ∈ {CommNode, StoreNode} do
 2     addVertex(node) ;
 3     foreach Comm | ∃ mapping (node,Comm)
         do
 4         addVertex(Comm);
 5         addEdge(node, Comm);
 6     end
 7 end
 8 foreach node in AM | node ∈ExecNode do
 9     addVertex(node);
10     foreach task | ∃ mapping (node,task) do
11         addVertex(task);
12         addEdge(node, task);
13         while ∃ operator θ | θ ∈ O_task do
14             previousOp.add({θ, sub_O});
15             if θ instanceof StartState then
16                 addVertex(θ );
17                 addEdge(task, θ );
18             else if θ instanceof StopState then
19                 addVertex(θ);
20                 foreach θ_i | θ_i ∈ previousOp
                     do
21                     if θ ∈ sub_Oi then
22                         addEdge(θ_i, θ);
23                         if θ_i ∈ sub_Oj | sub_Oj ∈
                             previousOp and θ_j
                             instanceof ForLoop
                             or Sequence then
24                             addEdge(θ, θ_j);
25                         end
26                         ...
27                 end
28             else
29                 addVertex(θ);
30                 foreach θ_i | θ_i ∈ previousOp
                     do
31                     if θ ∈ sub_Oi then
32                         addEdge(θ_i, θ);
33                 end
34             end
35         end
36     end
37 end
38 foreach link in Links do
39     addEdge(link_start,link_end)
40 end
41 foreach taskComm in Comm do
42     addEdge(taskComm_send,taskComm_rec)
43 end
```

as mandatory if it corresponds to a node $node_\theta$ that has path from $node_{\theta_A}$ or a path to $node_{\theta_B}$.

In addition to the traces added to *onPathTransactions*, s'_p might contain transactions that are not on the path between $node_{\theta_A}$ and $node_{\theta_B}$ but which may be executed on the same hardware as either θ_A or θ_B. These transactions, saved in *hardwareDelayTran* array, might contribute to λ_i because of the scheduling policies: they might delay transactions related to *onPathTransactions* thus contributing to an increase in λ_i.

TTool can display the result of algorithm 2 with a table. In this table, each row corresponds to one hardware node in the system and each column represents one time slot in the simulation. Transactions are placed according to when and where they were executed. The transactions that belong to *onPathTransactions* are displayed in green and the ones belonging to *hardwareDelayTran* are displayed in red, thus giving an immediate view of the simulation transactions and their related SysML elements involved in an extra delay. In the scope of this paper, the transactions in *hardwareDelayTran* are colored red regardless if their scheduling increased the delay between $startTime_{st_{i_A}}$ and $endTime_{st_{i_B}}$ or not. We intend to address this limitation (future work). Also, our algorithm does not (yet) handle extra delays due to contentions on communication and storage nodes.

5 CASE STUDY

A motor drive system —defined in the scope of the H2020 AQUAS project (AQU, 2013)— shows our performance analysis contribution from SysML models. The system consists of 3 main components: a motor, a motor controller and a client application. The motor controller receives speed and direction values from the client application and accordingly generates the right PWM (Pulse Width Modulation) signals and sends them to the motor. In addition, the motor controller regularly monitors the position data from the motor to accordingly adjust PWM signals as needed. Two requirements are considered in the paper:

1. A maximum delay between a user setting speed and direction values in the client application and time at which the motor controller takes these new values into consideration (safety requirement)
2. The speed and direction values sent from the client application to the controller should remain confidential (security requirement)

Algorithm 2: Simulation trace analysis.

Data: θ_A,θ_B, $startTime_{st_{i_A}}$,$endTime_{st_{i_B}}$, s_p , g
Result: Detailed time analysis between θ_A,θ_B

```
foreach SimulationTransaction st in s_p do
    if st.startTime >= startTime_{st_{i_A}}
    and st.endTime =< endTime_{st_{i_B}} then
        s'_p.add(st)
        if ∃ path(node_{θ_A} → node_{θ_B} ) in g
         then
            if node_{θ_st} ∈ path then
                onPathTransactions.add(st)
                 break;
            else if st.deviceName ==
             hardware_{θ_A||θ_B} then
                hardwareDelayTran.add(st);
        end
        else
            if ∃ path(node_{θ_A} → node_{θ_st} ) || ∃
             path( node_{θ_st} → node_{θ_B} ) then
                onPathTransactions.add(st)
            else if st.deviceName ==
             hardware_{θ_A||θ_B} then
                hardwareDelayTran.add(st);
        end
    end
end
```

5.1 HW/SW Partitioning Models

The functional model consists of 3 main tasks (Figure 3): *Client_App*, *Motor_Control_App*, and *Motor*. The *Client_App* gets speed and direction from a user and sends them to the *Main_Loop* in *Motor_Control_App* through the *Server_Control_App*. In each periodic iteration of *Main_Loop*, the speed and direction of the motor are deduced by reading the position data and current value sent from the *Motor* via *Motor_Control_Power*. Then the deduced values are compared with the required values of both the speed and direction. In case an adjustment is required, the *Main_Loop* sends updated PWM signals to the *Motor_Control_Power*. *Motor_Control_Power* transforms these signals into supply voltages and issues these signals to the motor.

Due to limited space, Figure 4 shows only *Client_App* and *Server_Control_App* applications and details the activity diagrams of only one task of each: *Update_MotorData* and *ReceiveMotorSD_Control*, shown on the left and right in Figure 4 where start and stop operators are colored black, request operators are in brown, event operators are in purple and action operators are in green. The functional tasks in Figure 4 are mapped on the architecture shown in Figure 5, with two CPUs (blue) and two memories (green) connected by buses (brown) and bridges (orange).

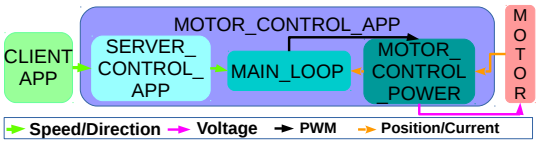

Figure 3: Case Study Overview.

5.2 System Verification

A simulation trace showing the latency between *sending a new input* in the *Client_App* (our operator "A" i.e. θ_A) and *receiving this input* in *Main_Loop* (operator "B" i.e. θ_B) is computed. The Latency Analysis tool of TTool takes as input this trace, the two operators (θ_A,θ_B), and the model so as to generate the latency graph. The latency graph of *Receive-MotorSD_Control* is given in Figure 6. The dotted edges in Figure 6 are logical dependency edges from/towards vertexes that are not shown in the graph. The latency detailed analysis shows that the time between θ_A and θ_B is 273 cycles. Figure 7 displays the detailed latency of the simulation trace. As mentioned earlier, the green colored cells in Figure 7 are the transactions corresponding to nodes ($node_\theta$) that are on the path from to θ_B, while those colored red are transactions not on the path but executed on the same hardware as θ_A or θ_B and might add delay due to scheduling.

To prove that the confidentiality requirement between *Client_App* and *Server_Control_App* is satisfied, TTool relies on ProVerif. In a first verification run the proof fails (as indicated by the crossed red lock in Figure 4). To address confidentiality, encryption/decryption operators are added as described in (Li, 2018). We chose the AES algorithm and set the computational complexity to 3000 as indicated in (Fujdiak et al., 2019). The verification now outputs a green lock in Figure 4: confidentiality is satisfied.

The latency detailed analysis for the secure model tells us that the time between θ_A and θ_B is now 6301 cycles. Figure 8 shows that the transactions that were executed during this delay start differing from the non secure model at cycle 11 where the encryption procedure starts. Studying the two latency detailed analysis figures allows us to conclude that the increase in time between the secure and non secure model is due to the added encryption/decryption. Moreover, the latency detailed analysis helps us to understand when the encryption/decryption started/ended and how did

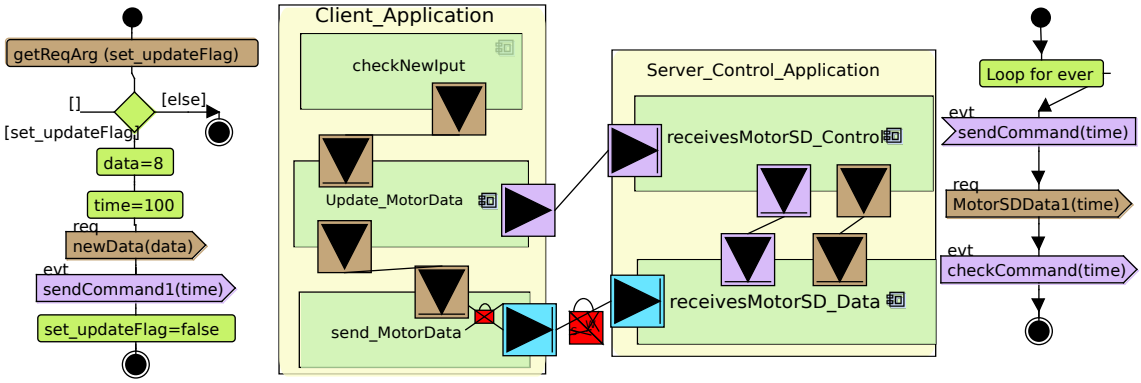

Figure 4: Functional View.

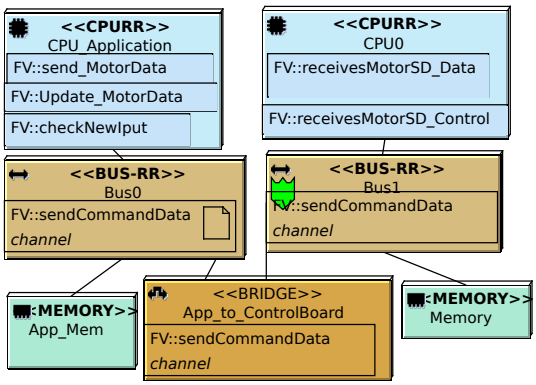

Figure 5: Mapping View.

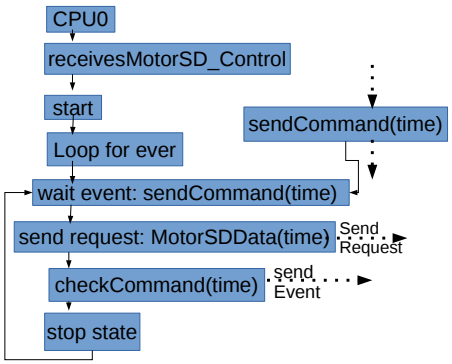

Figure 6: Excerpt of the Latency Graph.

it effect the scheduling of tasks.

In case the latency violated safety or performance requirements, the system designer can either add hardware accelerators for encryption/decryption, or more powerful processing units, or use other security algorithms, or try a different mapping, or finally adjust the scheduling policy of a CPU.

6 CONCLUSION AND PERSPECTIVES

The semi-automated performance analysis technique introduced in this paper allows system designers to study the trade-off between security and performance at high level of abstraction. To do so, simulation traces are analyzed based on a generated directed graph that corresponds to the model under investigation.

This work was integrated in SysML-Sec and TTool. A use case from the AQUAS H2020 project was used to highlight the advantages of knowing the transactions causing delays between two operators.

The presented approach could probably be extended to also cover safety aspects in a combined safety/security/performance holistic co-analysis. This extension will be the objective of future works

ACKNOWLEDGEMENTS

The AQUAS project is funded by ECSEL JU under grant agreement No 737475

REFERENCES

(2013). Aggregated quality assurance for systems (aquas). https://aquas-project.eu. Accessed: 2019-09-24.

Apvrille, L. (2013). TTool. https://ttool.telecom-paris.fr. Accessed: 2019-09-24.

Apvrille, L. and Li, L. W. (2019). Harmonizing safety, security and performance requirements in embedded systems. In *Design, Automation and Test in Europe (DATE'2019)*, Florence, Italy.

Apvrille, L. and Roudier, Y. (2013). SysML-Sec: A SysML environment for the design and development of secure

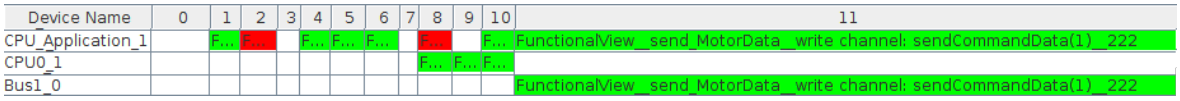

Figure 7: Detailed Latency between θ_A and θ_B.

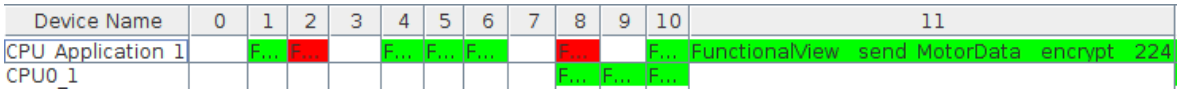

Figure 8: Detailed Latency between θ_A and θ_B in Secure Model.

embedded systems. *APCOSEC, Asia-Pacific Council on Systems Engineering*, pages 8–11.

Blanchet, B., Smyth, B., Cheval, V., and Sylvestre, M. (2018). Proverif 2.00: Automatic cryptographic protocol verifier, user manual and tutorial. *Version from*, pages 05–16.

Chai, H., Zhang, G., Zhou, J., Sun, J., Huang, L., and Wang, T. (2019). A short review of security-aware techniques in real-time embedded systems. *Journal of Circuits, Systems and Computers*, 28(02):1930002.

Davis, R. I. (2014). A review of fixed priority and edf scheduling for hard real-time uniprocessor systems. *ACM SIGBED Review*, 11(1):8–19.

DeAntoni, J. and Mallet, F. (2012). Timesquare: Treat your models with logical time. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 34–41. Springer.

Friedenthal, S., Moore, A., and Steiner, R. (2014). *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann.

Fujdiak, R., Blazek, P., Apvrille, L., Martinasek, Z., Mlynek, P., Pacalet, R., Smekal, D., Mrnustik, P., Barabas, M., and Zoor, M. (2019). Modeling the trade-off between security and performance to support the product life cycle. In *2019 8th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–6. IEEE.

Fujdiak, R., Mlynek, P., Blazek, P., Barabas, M., and Mrnustik, P. (2018). Seeking the relation between performance and security in modern systems: Metrics and measures. In *2018 41st International Conference on Telecommunications and Signal Processing (TSP)*, pages 1–5. IEEE.

Gruber, T., Schmittner, C., Matschnig, M., and Fischer, B. (2018). Co-engineering-in-the-loop. In *International Conference on Computer Safety, Reliability, and Security*, pages 151–163. Springer.

Jiang, W., Guo, Z., Ma, Y., and Sang, N. (2013). Measurement-based research on cryptographic algorithms for embedded real-time systems. *Journal of Systems Architecture*, 59(10):1394–1404.

Kangas, T., Kukkala, P., Orsila, H., Salminen, E., Hännikäinen, M., Hämäläinen, T. D., Riihimäki, J., and Kusilinna, K. (2006). Uml-based multiprocessor soc design framework. *ACM Transactions on Embedded Computing Systems (TECS)*, 5(2):281–320.

Kienhuis, B., Deprettere, E. F., Van der Wolf, P., and Vissers, K. (2001). A methodology to design programmable embedded systems. In *International Workshop on Embedded Computer Systems*, pages 18–37. Springer.

Knorreck, D. (2011). *UML-based design space exploration, fast simulation and static analysis*. PhD thesis, Telecom ParisTech.

Kocher, P., Lee, R., McGraw, G., Raghunathan, A., Moderator-Ravi, S., and Moderator-Ravi, S. (2004). Security as a new dimension in embedded system design. In *Proceedings of the 41st annual Design Automation Conference*, pages 753–760. ACM.

Li, L. (2018). *Approche orientée modèles pour la sûreté et la sécurité des systèmes embarqués*. PhD thesis, Paris Saclay.

Li, L. W., Lugou, F., and Apvrille, L. (2017). Security-aware modeling and analysis for hw/sw partitioning. In *5th International Conference on Model-Driven Engineering and Software Development (Modelsward)*, Porto, Portugal.

Thiele, L., Wandeler, E., and Haid, W. (2007). Performance analysis of distributed embedded systems. In *International Conference On Embedded Software: Proceedings of the 7 th ACM & IEEE international conference on Embedded software*, volume 30, pages 10–10. Citeseer.

Viehl, A., Schönwald, T., Bringmann, O., and Rosenstiel, W. (2006). Formal performance analysis and simulation of uml/sysml models for esl design. In *Proceedings of the conference on Design, automation and test in Europe: Proceedings*, pages 242–247. European Design and Automation Association.