2017 12th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)

Securing C/C++ Applications with a SEcube™-based Model-driven Approach

Frederik Gossen
Lero - The Irish Software Research Centre
University of Limerick, Ireland
Email: frederik.gossen@lero.ie

Johannes Neubauer*, Bernhard Steffen†
TU Dortmund University, Germany
Email: *johannes.neubauer@tu-dortmund.de,
†bernhard.steffen@tu-dortmund.de

***Abstract*—In this paper we demonstrate the power and flexibility of extreme model-driven design using C-IME, our integrated modelling environment for C/C++ by showing how easily an application modelled in C-IME can be enhanced with hardware security features. In fact, our approach does not require any changes of the application model. Rather, C-IME provides a dedicated modelling language for code generators which embodies a palette of security primitives that are implemented based on the SEcube™ API. We will illustrate how the required code generator can be modelled for a to-do list management application in our case study. It should be noted that this code generator is not limited to the considered application but it can be used to secure the file handling of any application modelled in C-IME.**

***Index Terms*—SEcube™, Extreme Model-driven Design, Security, Full Code Generation, Service Orientation.**

I. Introduction

In a connected world security becomes more and more important as attacks emerge quickly on the wealth of new services [1]. Security concerns are often dealt with already early in the development process. However, many security requirements become known much later, often even after deployment [1], [2]. Thus methods are required that allow to security-enhance applications which are already fully functional.

In this paper we demonstrate the power and flexibility of extreme model-driven design (XMDD) [3], [4] by showing how easily an application modelled in our integrated modelling environment for C/C++, C-IME, can be enhanced with hardware security features without even touching its models. C-IME is a product variant in a product line of integrated modelling environments[1] (IMEs) built with the meta tooling suite Cinco [5]. Hardware security is included based on SEcube™ security technology [6], an open security platform that provides encryption running on a separate hardware device.

Key to our approach is C-IME's dedicated graphical modelling language for code generators which embodies a palette of security primitives that are implemented on top of the SEcube™ APIs as the highest level of abstraction in the SEcube™ software collection (cf. Fig. 1). These code generators generate code from their argument application models fully automatically. Their structure is simple and consists of three parts (cf. Sec. V):

- An initialization part generates setup code for the target platform. In our case study, this mainly concerns setting up the SEcube™ device and performing the login procedure.
- The code generation for the application model. In the modelling language for code generators a mapping from the modelling level to the implementation level in C is defined that allows to generate the same model differently using different code generators. In our case study it is important to ensure secure file operations using SEcube™-based security primitives.
- A counterpart to the first step that generates code to clean up when the application terminates. This is where the logout from the SEcube™ is performed.

We will illustrate how easily the required code generator can be modelled for a secure to-do list management application in our case study. However, this code generator is by no means limited to this application and it can be used to secure the file handling of any application modelled in C-IME.

In Sections II and III we introduce the reader to the modelling environment C-IME and the process of generating applications from their models. Section IV describes the SEcube™ hardware security technology and its API, utilized in the case study in Section V. Our findings and an outlook on future work conclude in Section VI.

II. Modelling Environment

The modelling environment C-IME is a product in a product-line of application modelling environments all created with Cinco. A very advanced product in this product-line is Dime (*dynamic web application integrated modelling environment*) tailored to model web applications which can be one-click generated into a Java EE web stack fully automatically. The website of the SEcube™ community, e.g., is completely modelled in Dime [8].

Cinco-products are based on meta models defining a set of *nodes* and *edges*, their appearance, and constraining the way they can be connected. These definitions are shared between Dime and C-IME for process models, also called *service logic*

[1]We call development environments for (graphical) modelling languages IMEs reminiscent to integrated development environments (IDEs) for classical programming like Eclipse or IntelliJ.

978-1-5090-6377-2/17/$31.00 ©2017 IEEE

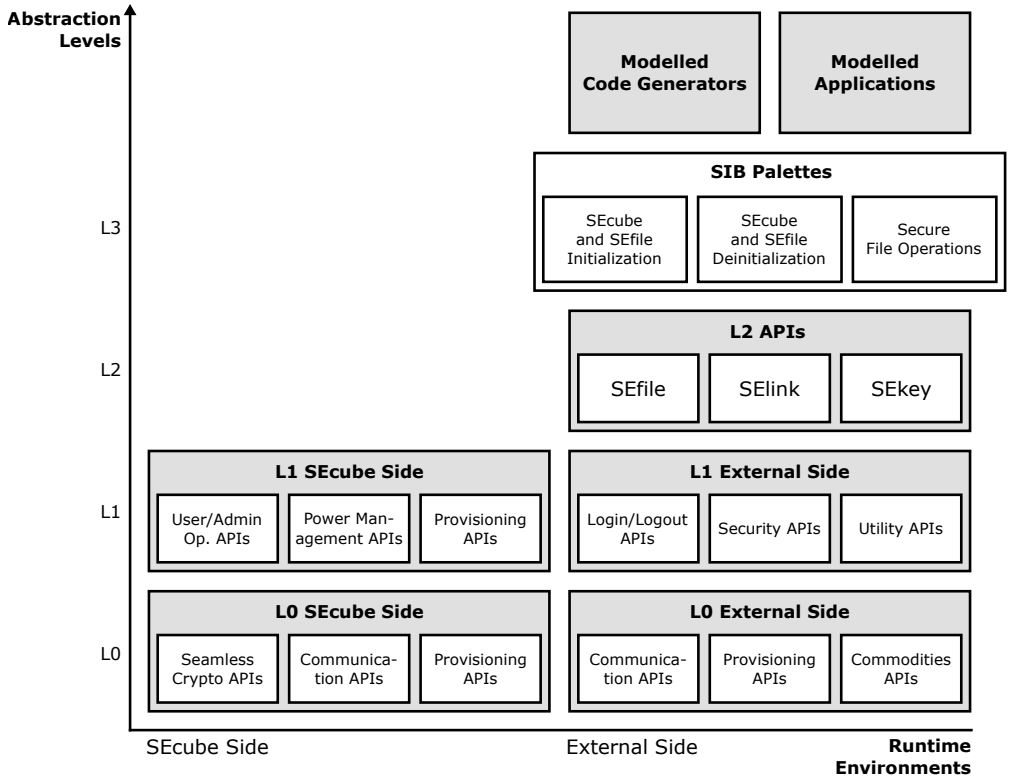

Fig. 1: Overview of the SEcube™ software collection [7]

graphs (SLGs). While nodes on the graph canvas represent activities, different types of edges model control flow and data flow. Figure 2 shows an example of a C-IME model in the to-do list management application.

The central types of nodes are called *service independent building blocks* (SIBs) which denote activities in the model (e.g., `PrintString` in Figure 2). A SIB has a list of typed *input ports* (cf. `todos` and `todosFinished` nodes in the SIB labelled `start`) similar to function or method parameters in general-purpose programming languages. The functionality represented by the SIB relies on the data provided by means of these input ports. The execution of a SIB can result in different cases of outcome which are modelled using the concept of *branches*. SIBs have one or more branch nodes as their successors, each representing one case (e.g., nodes labelled `true`, `false` in Figure 2). Depending on the outcome of the execution of a SIB, one branch is chosen, determining the successor SIB to be executed next. In this way branches are used to model the control flow of the system (cf. solid edges in Figure 2). Each branch may provide arbitrary output via typed *output ports*.

The unique entry point of an SLG is the *start SIB* which is represented by a dedicated node type labelled `start`. A start SIB may define output ports like a branch that define the counterpart to formal parameters in a method declaration. Further on, an SLG has one or more exit points, the *end SIBs* (cf. node labelled `done` in Figure 2). The latter may define arbitrary input ports as each end SIB defines a case of outcome of the process. Together, start and end SIBs declare the interface of a process model. Consequently, a process model may be integrated into another SLG as a *process SIB* with input ports, branches and output ports (i.e., the actual parameters) corresponding to the declaration in the sub process' start and end SIBs, thus introducing hierarchical modelling. Furthermore, the *native SIBs* represent calls to C functions, introducing service-orientation to C-IME. The mapping of a SIB interface to a method signature is done in a separate textual *domain-specific language*.

As C-IME allows to model both control flow and data flow, it provides different kinds of edges. Data can be supplied to SIBs via the initial output ports of the start SIB or it can be provided via the outcome of a previously executed SIB. An output port may be connected using *direct data flow* edges to one or more input ports (cf. the dashed grey edge from output port `todos` to input port `content` in Figure 2).

In any case, the type declarations of the output port and the connected input ports have to match. The types are denoted after the name of a port (e.g., `todos: [Text]`). Square brackets around a type name denote a list type. A model validation component performs type checking [9]. C-IME supports the set of *primitive types* `Text`, `Boolean`, `Integer`, `Real`, `Timestamp` as well as their *list types*. Additionally, *native types* can be used to represent arbitrary types in the target language. For input ports with primitive types a modeller may

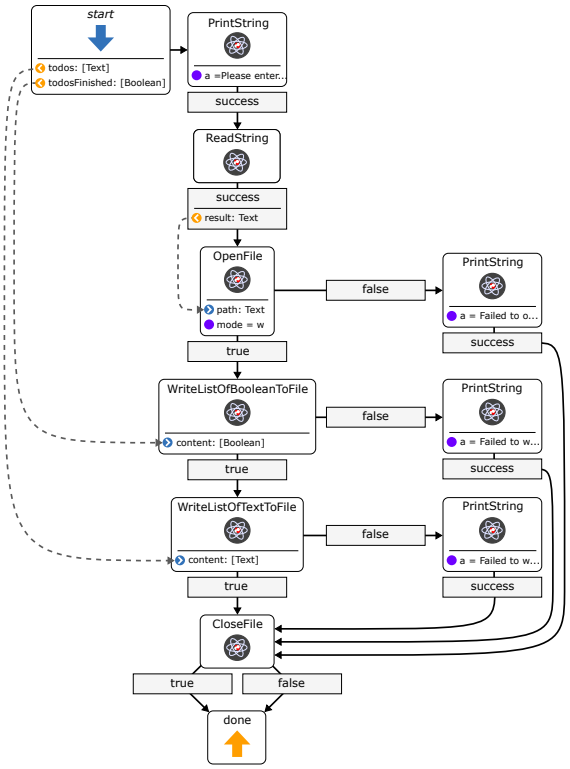

Fig. 2: C-IME model for writing the to-do list management application's data model to the file system

supply a static value instead of using data flow edges (cf. the parameter `a` of the SIBs `PrintString` in Figure 2).

III. Code Generation

A modeller in C-IME does not have to deal with any details of the underlying implementation. So the code generator has to take care of transferring the semantics of the models to C code. The target environments will in most cases be embedded systems on very constraint hardware, with limited computing power and available memory. Therefore, a very slim library has been implemented along with the generator. This library provides C counterparts for the types in C-IME as well as list types. Furthermore, the library offers a very simple memory management solution facilitating reference counting. This library is shipped together with C-IME by means of an easy to use project template.

The code generator in C-IME produces one `.c` and one `.h` file per SLG following the separate compilation principle [10]. Analogously, the aforementioned library code as well as all method calls for native SIBs base solely on the corresponding header files. This way, the implementation of all SIBs (including process SIBs) and library functions can be replaced transparently (cf. Sec. V).

In contrast to Java – the generation target of DIME – C does not support high-level language features like object-orientation, subtype polymorphism, garbage collection or even namespaces. Hence, all this is emulated in the generated code. Functions are disambiguated using unique names. The graph structure of a process cannot be represented canonically in a block-oriented language like C [11]. This is solved by iterating through an adjacency structure, avoiding a deep call stack. While the Java generator in DIME realizes SIB execution via calls to polymorphic `execute`-methods, this is emulated in C by function pointers to corresponding `execute`-functions. The generator renders dedicated implementations of these `execute`-functions for the various kinds of SIBs, which encapsulate the concrete execution as well as the handling of the data flow.

The direct data flow is realized via a struct that is passed between SIB executions. Branches of an SLG are analogously implemented as a struct with one member indicating the branch that was taken and one member for every branch holding an inner struct representing its output ports' values.

All necessary calls to library methods for the creation of constants and lists, for memory allocations and deallocations as well as for reference counting are generated directly into the C code.

IV. Hardware Security

While software implementations of encryption are cost-effective and flexible, implementation in hardware is generally regarded as being more secure. It is less vulnerable to common attacks of which a good summary can be found in [12] also discussing general advantages and disadvantages of software vs. hardware implementation of encryption.

A. SEcube™ Security Platform

One example for a hardware security platform is the SEcube™ [6]. The platform implements a set of commonly used encryption algorithms that run on a device separate from the host machine. In this way the encryption mechanism is less vulnerable to the attacks listed in [12] than implementations in software. In particular, the following properties tackle common security risks:

- *No Security Bound Imposed by the OS:* All encryption algorithms run on the SEcube™ device, that is separate from the host machine. This way they rely in no way on host machine's operating system and cannot be attacked through its potential security flaws.
- *No Arbitrary Memory Access:* The SEcube™ device has its own memory, that is physically separate from the host machine. It is therefore not accessible to other software running on the same machine.
- *Data Integrity Guarantee:* The firmware running on the SEcube™ device cannot be altered through its interfaces. While this makes the device more secure, it also makes the solution less flexible. Unlike software implementations of encryption, the SEcube™ firmware cannot be updated after the device was physically closed.
- *Secure Key Storage:* It is crucial to store keys used for encryption and decryption in a secure location. In case of the SEcube™, keys are securely stored on the device which they will never leave. Keys are identified by their

ID and can be used on the particular SEcube™ device only.

The SEcube™ platform is of particular interest to us because it is an entirely open platform. Most importantly, the software running on the device as well as the API to be included in applications on the host machine is open source. While the open source concept is common to software, it is not yet commonly applied to hardware especially in the security domain [6]. Besides the SEcube™ platform, there exist very few security-oriented open hardware platforms.

B. SEcube™ API

The SEcube™ comes with a set of APIs for the programming language C. It is organized in three layers, allowing security experts to interact with the device in a fine granular fashion, whereas non-security experts can use higher abstraction levels. On the highest level, the API allows to interact with the file system and have all content implicitly encrypted. This allows the user to take advantage of the technology without having to struggle with the details of encryption.

The lowest layer L0 of the API allows to interact with the SEcube™ device directly. It provides basic functionality to discover devices, to open and to close them and also to communicate with them through one generic function. While some functions are to be used by experts only, device discovery and open/close operations are useful to every user.

The second layer L1 of the API provides functionality to log into the device and to perform encryption and decryption of data on a byte level. The layer also allows to set up a connected SEcube™ device, meaning to add, delete and list keys, to set the user and the admin PIN and to list available algorithms.

The highest layer L2 of the API consists of SEfile™ and SElink™ which are used to encrypt data at rest respectively data in motion. The provided functions implement common interfaces used to interact with the file system and to send data over networks. As encryption happens implicitly, the user does not have to juggle with technical details of the SEcube™. In our case study, we utilize this to automatically secure file operations of an application modelled in C-IME.

V. Case Study: To-do List Management Application

To show the power and flexibility of extreme model-driven design, we study the case of a small to-do list management application that allows to keep track of activities to be done in the future. The application allows the user to add new items to a list and to keep track of their state, namely whether a task is finished or still pending. While this application is reasonable at the extent of its functionality it has a high demand for security in some use cases. A user wants to securely save his or her data to the hard drive and does not want it to be readable or alterable by unauthorized others.

The original to-do list management application was already modelled in C-IME prior to this work. Like many other applications, it uses read and write file operations to persist its data model to the hard drive. These operations are typically not encrypted. In order to secure the operations, it is desirable to use a technique that does not require any change of the application's models, to be generic enough to apply the very same technique to any other application modelled in C-IME. To this aim, we propose a method that instead models the C-IME code generator using yet other domain-specific languages in such a way that critical parts of any C-IME modelled application can be automatically secured during its code generation as part of an enhanced code generation process.

```
secube@dtis:~/secube/todoapp/build$ ./app
>> Please enter the user PIN
user
>> Do you want to 'add <name>', 'delete <num>', 'clear',
read
>> Please enter a filename.
ToDoFebruary
>> Do you want to 'add <name>', 'delete <num>', 'clear',
list
>> Buy Milk
>> Buy Cookies
>> Finish DTIS Paper
>> Do you want to 'add <name>', 'delete <num>', 'clear',
finish 2
>> Do you want to 'add <name>', 'delete <num>', 'clear'
list
>> Buy Milk
>> Buy Cookies
>> Finish DTIS Paper [done]
>> Do you want to 'add <name>', 'delete <num>', 'clea
write
>> Please enter a filename.
ToDoFebruary
>> Do you want to 'add <name>', 'delete <num>', 'cl
exit
>> See you later, alligator.
```

Fig. 3: Secure to-do list management application during execution

A. To-do Application Description

Although the basic functionality of a to-do list management application is generally known, we will describe the features included in this case study in detail before discussing security critical parts of the actual C-IME models. The fully generated to-do list management application is a command line tool that allows the user to manage all items on a to-do list by means of simple commands. The application further allows the user to write the list to a file and read it back. This can be done in a secure manner with encryption using the SEcube™ technology or in an unsecure manner using standard C functionality. These two alternatives will both be generated from the very same model, allowing the user to choose the one he or she prefers.

The to-do list management application takes commands as an input and processes them in a loop until the user terminates the application. Both versions of the application offer the same user commands:

- `add` *title* adds a new element with a given title to the to-do list. The element is initially added as unfinished.
- `finish` i sets the state of the i^{th} element to finished.

- `delete` i deletes the i^{th} element from the to-do list counting from zero.
- `clear` deletes all items from the to-do list.
- `list` prints all items and their states to the console.
- `write` allows the user to write the to-do list to a file. The user will be prompted for a filename.
- `read` allows to read a to-do list from a file. The user will be prompted for a filename and all items will be appended to the existing to-do list.
- `exit` terminates the application.

Figure 3 shows a screenshot of the to-do list management application during execution. In this secure version, the user is prompted for the PIN to log into the SEcube™ immediately after the application was started. The user reads a to-do list from a file, marks one item as finished and writes the list back to the file.

B. To-do List Management Application Model

The to-do list management application that was already modelled in C-IME prior to this work, but did not use encryption and the security requirement to encrypt all data when stored on the hard drive was stated only after the application was deployed. It is easy to identify security critical parts with regard to this security requirement as all C-IME applications are modelled based on a dedicated palette of SIBs. In case of the to-do list management application, the critical operations are `OpenFile`, `Write...ToFile`, `Read...FromFile` and `CloseFile`. The C-IME models for reading and writing the to-do list are similar, thus we will consider only the write process in what follows.

Figure 2 shows the sub model within the to-do list management application that writes its simple data model to the hard drive. The only data stored in memory at runtime are the to-do list's items. They consist of a title and a Boolean state indicating whether or not the item was already marked as finished. In the application's data model this is represented by two lists, one of texts and one of Boolean values. Only these two lists are given to the write process as the input parameters `todos` and `todosFinished`.

Before writing anything, the to-do list management application needs a filename to identify a file to write to. When starting the execution of the sub model, the first SIB executed, `PrintString`, prompts the user to enter a filename. The SIB is given a static text as an input and prints it to the console telling the user what to do. The only possible outcome of the SIB is `success` with no data associated with it. To read the user's choice from the command line, the SIB `ReadString` is used. Again, this SIB has `success` as its only possible outcome but it provides the desired filename as a result through an output port. With this information, the SIB `OpenFile` can open the file. It takes the filename as an input directly from the output port of its predecessor. In contrast to the first SIBs, file operations can fail for many reasons. This is modelled by means of a Boolean return value indicating whether or not the operation was successful. In C-IME models these two possible outcomes are represented as two branches, `true` and `false`, enabling the modeller to split up the control flow. While in case of success, the model proceeds and writes the data model to the open file, in case of failure it prints an error message and aborts the process. If all file operations succeed, the execution will trace vertically from the top down, opening the file, writing both lists to it and finally closing it. The lists are passed from the sub models input parameters to the SIB's input ports through direct data flow edges. In case any of the file operations fail, the model will reach the corresponding `PrintString` SIB which prints a meaningful static error message before the sub model terminates.

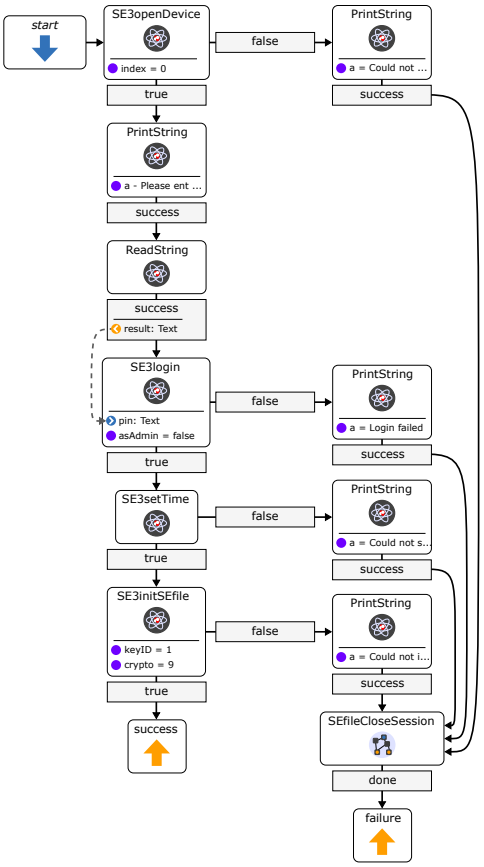

Fig. 4: C-IME model for setting up the SEcube™ environment to allow for secure file operations

Note that both read and write file operations are available for all the primitive types and lists supported by C-IME. This includes `Text`, `Boolean` and also lists of these types, allowing the modeller to seamlessly persist the to-do list management application's data model to the file system.

C. Modelling a Secure Code Generator

All SIBs used in the to-do list management application can be realized by any implementation that conforms to their interfaces (cf. Sec. III). This allows to generate different versions of an application from the exact same C-IME model. In particular, it allows to generate both, a secure and an unsecure version of the to-do list management application from its model using

customized code generators. Only the operations `OpenFile`, `Read...FromFile`, `Write...ToFile` and `CloseFile` have to be generated with different implementations.

As described in Section III, implementations of native SIBs are not included but only invoked from within the generated code. The expected implementations must therefore be available when compiling the generated code. This is where a customized code generator may choose one of multiple implementations to automatically secure file operations. Such a customized code generator is independent of the application model and can thus be used to secure not only the to-do list management application but any application modelled in C-IME.

In order to allow for the use of hardware encryption by means of the SEcube™ security technology, two additional steps are necessary, namely to initialize the SEcube™ API and to finalize it. These two steps can be performed at the very beginning respectively at the very end of any application's execution so that they can become a part of the enhanced code generator itself. In this way the actual application's models remain untouched and security requirements are automatically implemented by the code generation process.

To model a code generator for C-IME, additional domain-specific languages are used to define the following three parts:

- An initialization process to set up the environment for the SIB implementations that are included in the generated application. This can be the login routine for the SEcube™, where the device is opened and the user authenticates him or herself. The initialization process is modelled using SLGs similarly to the main application, but with a different SIB palette dedicated to initialize the SEcube™. The modelled process will be automatically included in any generated application by this particular code generator.
- A mapping from SIBs in the model to concrete implementations written in C. Depending on this mapping, the code generator will choose the secure or the unsecure version of file operations.
- A finalization process is the counterpart to the initialization to clean up the environment. The logout procedure from the SEcube™ device happens here.

In case of the to-do list management application, the secure and the unsecure version are generated by two different code generators, both modelled in our domain-specific languages. With the exception of file operations, both code generators use the same mapping from SIBs to implementations. While the unsecure code generator will include an implementation based on the standard C file operations, the secure version of the code generator chooses an implementation based on the SEfile™ API.

Initialization and finalization are only needed to set up the SEcube™ environment. For the code generator that generates unsecure applications the two processes are empty, while the secure version of the code generator includes modelled login and logout processes. Figure 4 shows the model for the secure code generator's initialization. The model invokes a sequence of native SIBs each of which can succeed or fail. In case of failure an error message is printed to inform the user and the model is immediately terminated. To enable the SEfile™ API used by the SIB's secure implementations, the SEcube™ is opened, the user is prompted for the PIN and the login is finally performed. To use encryption, it is further necessary to set the time and encryption parameters. The model for finalization is even less complicated keeping all models for the secure code generator small and simple.

We thus obtain a code generator that is not only very flexible itself but also generates any application modelled in C-IME with implicitly secured file operations. Note that the code generator is in no way specific to the to-do list management application but it can easily be applied to any other application as well. By modelling not only the application but also the code generator, we are not even limited to SEcube™ security technology. In fact, we can wrap any other implementation for secured operations allowing also for software encryption if needed.

VI. Conclusion

In this paper we have shown how to easily enhance running C/C++ applications developed within our C-IME environment with hardware secured file handling simply by changing the according code generator. Key to our approach is the full code generation philosophy of XMDD, which guarantees that each running application has a valid model from which it was generated. This approach allows us to realize certain cross-cutting concerns like security via a corresponding code generator in a fashion reminiscent of aspect oriented programming [13] without touching its application models. C-IME supports this approach by providing a dedicated graphical modelling language for code generators which embodies a palette of security primitives that are implemented using the SEcube™, a hardware device for enhanced security. We have illustrated how easily such code generators can be modelled using a to-do list management application as a case study. It should be noted that handling security via the code generator this way is not application specific but allows us to automatically secure the file handling of any application modelled in C-IME.

Currently, we only secure applications concerning data at rest. The philosophy underlying the XMDD paradigm [14], where complex applications are modelled via a number of strongly linked individual models is however much more general, as witnessed, e.g., by DIME [9], [15], [16] our most elaborate CINCO-product variant. We are currently investigating how to best approach also aspects like data in motion in this context. Additionally, up to now we introduced solely compile-time variability for C-IME, although we have shown the impact of runtime variability for a wide range of applications [17], [18] for high-level programming language targets. We are planning to apply this approach also to C-IME. The to-do list management application, e.g., would significantly benefit if the application could dynamically decide at runtime to use hardware security in case the SEcube™ device is available, and degrade to a software security variant as long

as a cryptographic key is known, or else simply use an unsecured variant. Important for us is, however, the simplicity principle [19], [20], which clearly favours reduced tailored solutions to generic approaches. In this case this means for a modeller to be able to model any C-IME application without worrying about security concerns, which adds an incremental requirement specification flavour to the entire design approach [21].

ACKNOWLEDGMENT

This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie).

REFERENCES

[1] G. Elahi, E. Yu, T. Li, and L. Liu, "Security Requirements Engineering in the Wild: A Survey of Common Practices," in *2011 IEEE 35th COMPSAC*, pp. 314–319, July 2011.

[2] P. T. Devanbu and S. Stubblebine, "Software Engineering for Security: A Roadmap," in *FOSE Proceedings*, ICSE'00, (New York, NY, USA), pp. 227–239, ACM, 2000.

[3] T. Margaria and B. Steffen, "Agile IT: Thinking in User-Centric Models," in *ISoLA'08 Proceedings* (T. Margaria and B. Steffen, eds.), (Berlin, Heidelberg), pp. 490–502, Springer, 2008.

[4] T. Margaria and B. Steffen, "Service-Orientation: Conquering Complexity with XMDD," in *Conquering Complexity* (M. Hinchey and L. Coyle, eds.), (London), pp. 217–236, Springer, 2012.

[5] S. Naujokat, M. Lybecait, D. Kopetzki, and B. Steffen, "CINCO: A Simplicity-Driven Approach to Full Generation of Domain-Specific Graphical Modeling Tools," *Software Tools for Technology Transfer*, 2017. to appear.

[6] A. Varriale, E. I. Vatajelu, G. D. Natale, P. Prinetto, P. Trotta, and T. Margaria, "SEcube^{TM}: An Open-Source Security Platform in a Single SoC," in *DTIS Proceedings*, pp. 1–6, April 2016.

[7] G. A. Farulla, P. Prinetto, and A. Varriale, "Holistic security via complex hw/sw platforms," in *this volume*, 2017.

[8] S. Boßelmann, D. Kühn, and T. Margaria, "A fully model-based approach to the design of the secube^{TM}community web app," in *this volume*, 2017.

[9] S. Boßelmann, M. Frohme, D. Kopetzki, M. Lybecait, S. Naujokat, J. Neubauer, D. Wirkner, P. Zweihoff, and B. Steffen, "DIME: A Programming-Less Modeling Environment for Web Applications," in *ISoLA'16 Proceedings*, vol. 9953 of *LNCS*, pp. 809–832, Springer, 2016.

[10] D. Ancona, G. Lagorio, and E. Zucca, "A Formal Framework for Java Separate Compilation," in *ECOOP Proceedings*, ECOOP '02, (London, UK, UK), pp. 609–636, Springer-Verlag, 2002.

[11] E. Engeler, "Structure and Meaning of Elementary Programs," in *Symposium on Semantics of Algorithmic Languages*, pp. 89–101, Springer, 1971.

[12] N. Sklavos, K. Touliou, and C. Efstathiou, "Exploiting Cryptographic Architectures over Hardware vs. Software Implementations: Advantages and Trade-Offs," in *AEE'06 Proceedings* (D. Biolek, ed.), (Stevens Point, Wisconsin, USA), pp. 147–151, WSEAS, 2006.

[13] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-Oriented Programming," in *ECOOP'97* (M. Akşit and S. Matsuoka, eds.), vol. 1241 of *LNCS*, pp. 220–242, Springer, 1997.

[14] T. Margaria and B. Steffen, "Business Process Modelling in the jABC: The One-Thing-Approach," in *Handbook of Research on Business Process Modeling* (J. Cardoso and W. van der Aalst, eds.), IGI Global, 2009.

[15] S. Boßelmann, J. Neubauer, S. Naujokat, and B. Steffen, "Model-Driven Design of Secure High Assurance Systems: An Introduction to the Open Platform from the User Perspective," in *SAM'16. Special Track "End-to-end Security and Cybersecurity: from the Hardware to Application"* (T.Margaria and A. M.G.Solo, eds.), pp. 145–151, CREA Press, 2016.

[16] F. Gossen, T. Margaria, and T. Göke, "Modelling the People Recognition Pipeline in Access Control Systems," in *Proceedings of the Institute for System Programming*, vol. 28, pp. 205–220, 2016.

[17] J. Neubauer and B. Steffen, "Plug-and-Play Higher-Order Process Integration," *IEEE Computer*, vol. 46, pp. 56–62, August 2013.

[18] J. Neubauer and B. Steffen, "Second-Order Servification," in *Software Business. From Physical Products to Software Services and Solutions* (G. Herzwurm and T. Margaria, eds.), vol. 150 of *LNBIP*, pp. 13–25, Springer, 2013.

[19] T. Margaria, B. D. Floyd, and B. Steffen, "IT Simply Works: Simplicity and Embedded Systems Design," in *IEEE 35th COMPSACW*, pp. 194–199, July 2011.

[20] T. Margaria and B. Steffen, "Simplicity as a driver for agile innovation," *Computer*, vol. 43, pp. 90–92, June 2010.

[21] B. Jonsson, T. Margaria, G. Naeser, J. Nyström, and B. Steffen, "Incremental requirement specification for evolving systems," *Nordic J. of Computing*, vol. 8, pp. 65–87, Mar. 2001.