# Experience Report on the Development of a Specialized Multi-view Multi-stakeholder Model-Based Engineering Framework

Gurvan Le Guernic

# Experience Report on the Development of a Specialized Multi-view Multi-stakeholder Model-Based Engineering Framework

Gurvan Le Guernic
DGA Maîtrise de l'Information
Inria, Univ Rennes, CNRS, IRISA
Rennes, France

## Abstract

This paper reports on industrial experimentations to develop a dedicated model-based framework (process and tool support) aimed at supporting a subset of an existing document-based engineering process involving different teams belonging to two main stakeholders. The process supported covers the design of security related products by a prime contractor, and the supervision of this work by the contracting authority. This paper provides more details on this process, the requirements for the model-based engineering framework, and the work achieved. It then discusses the first results obtained from those experimentations, the lessons learned, and provides feedback for future similar works. Those relate mainly around: the benefits of an explicit and detailed methodology; the customizability of a general modeling language such as SysML for the development of a specialized model-based framework; and the importance of the distinction and clear definition of the problem domain, which correspond to the semantics of the models, and the solution domain, which corresponds to the syntax of the models.

***CCS Concepts*** • **General and reference** → *Experimentation*; • **Computing methodologies** → **Modeling methodologies**; • **Applied computing** → Computer-aided design; • **Software and its engineering** → Model-driven software engineering; *System modeling languages.*

***Keywords*** Model-based engineering, experience report, framework development, customization

**ACM Reference Format:**
Gurvan Le Guernic. 2019. Experience Report on the Development of a Specialized Multi-view Multi-stakeholder Model-Based Engineering Framework. In *Proceedings of the 17th ACM SIGPLAN International Workshop on Domain-Specific Modeling (DSM '19), October 20, 2019, Athens, Greece.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3358501.3361237

*DSM '19, October 20, 2019, Athens, Greece*


## 1 Introduction

The experiences reported in this paper fit in the specific context where a stakeholder (contracting or certification authority) interacts with other stakeholders (prime contractors or product vendors) to ensure that a product meets the precise and complex requirements of the first stakeholder (contracting or certification authority). In the security domain and for example, this setting is encountered in procurement processes, NATO certifications, FIPS-140's Cryptographic Module Validation Program (CMVP), or more commonly Common Criteria (CC) certifications [25]. It can also be encountered in other domains such as avionics or nuclear safety. Requirements on information pieces, needed by the authority stakeholder to be able to do its work, are usually explicitly defined. Nowadays and partly due to the fact that one authority stakeholder may have to work with different contractors or vendors that apply different processes and use different tools, those information pieces are mainly exchanged through numerous textual technical documents, which can be ambiguous and sometimes inconsistent [5, 16].

The experiments reported in this paper aim at transforming parts of a specific document-based process involving a contracting authority and a prime contractor (described in sect. 2.1) into a model-based process in order to facilitate and automate the validation of the information pieces exchanged and their evaluation. These experiments aim at developing a *Specialized Model-based Engineering Framework* (SMEF) for such a technical sub-process. It is *specialized* in the sense that it facilitates the engineering tasks of this specific sub-process, it is not aimed to be a general purpose model-based engineering framework. It is a *framework* in the sense that it is composed of a specific methodology and its supporting tools. The methodology relies on a dedicated SysML [21] profile, and the tools are built on top of Eclipse [26] and Papyrus [12, 27].

The paper focuses mainly on the context, the findings and personal thoughts resulting from the experimentation, rather than on the specialized model-based engineering framework

itself. The main lessons learned from this experimentation relate to the following points:

- the importance (for the development and use of the modeling framework) of an explicit and detailed definition of the process supported and the information manipulated by this process;
- the importance of a dedicated and customized supporting tool;
- the difficulty to develop such a dedicated tool;
- and the difficulty to embed a dedicated meta-model for a complex set of information pieces into a SysML profile.

The next section details the context of the work. It describes the sub-process for which the framework (SMEF) is developed and the objectives that were to be met. Section 3 describes succinctly the work that has been done, while sect. 4 present the raw facts that resulted from the experiments. Then the paper generalizes those facts in sect. 5 and gives personal thoughts and opinions on approaches to improve on such works, before concluding in sect. 6.

## 2 Detailed Context and Objectives

This section describes first the context of deployment of the future SMEF, and then explicits the main requirements that the SMEF has to fulfill.

### 2.1 Context of Deployment

Figure 1 presents the general process into which this work fits. This process is a generic procurement process for specific products that provides IT security related services. In the remainder of this paper, following the CC [25] terminology, the product acquired during this process is called the *Target of Evaluation* (ToE). Among other things, the goal of the close supervision by the contracting authority (CA) of the work of the prime contractor (PC) is to ensure, as much as possible, that the product's design respects some good practices and has good properties that allows for a smooth evaluation phase. This fine-grained early-stage evaluation of the product design requires that both stakeholders (CA and PC) communicate using specific and detailed information on the product's requirements and envisioned architecture. The information exchanged and the level of details required have been explicitly defined by the CA, and the PC is required to deliver specific (textual) documents that respects those requirements. The sub-process of interest for this paper, and for which a specialized model-based engineering framework (SMEF) is developed, resides in the dashed area (with other sub-processes). Communication in this sub-process is currently mainly based on 2 textual documents (that are also used by other sub-processes): a document describing the security requirements the product meets (called CdS); and a document describing the envisioned architecture (called DEC).

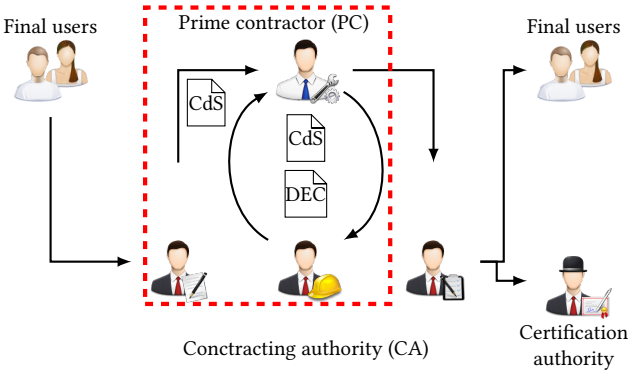



**Figure 1.** Acquisition Process for IT Security Products

The CdS is similar to the *Security Target* document of the *Common Criteria for Information Technology Security Evaluation* (CC) [14, 25]. The CdS is based on the result of a security analysis of the ToE. It includes a black box definition of the product identifying, among other information: the environment of the ToE (actors and other devices in interaction with the ToE); its interfaces; the services provided by the ToE; and which actors have access to those services. Most importantly, it also includes a description of the security related features and objectives that the ToE must achieve.

The DEC shares some similarities with other artifacts of the CC. It complements with more details the black box definition of the ToE, and provides two white box specifications of the ToE: an "implementation independent" architecture that focuses on the functional behavior and decomposition of the ToE and shares similarities with a *Platform Independent Model* (PIM) [7, 22]; and an "implementation focused" architecture which emphasizes the specific hardware choices and shares similarities with a *Platform Specific Model* (PSM) [7, 22].

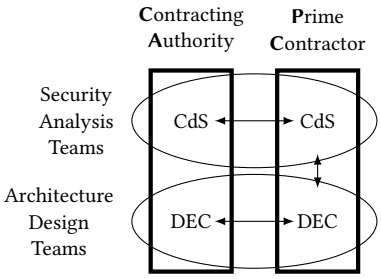



**Figure 2.** Exchanges overview

As illustrated in Fig. 2, during the sub-process of interest for this paper, and for which a specialized model-based engineering framework (SMEF) is developed, the CdS is first initialized by the contracting authority (CA) and then updated regularly by the prime contractor (PC). The DEC goes through an iterative process of redaction by the PC and evaluation by the CA. Along with the refinement of the design of the ToE, taking into account information in the CdS, more and more details are added to the DEC, that may in turn impact the CdS.

### 2.2 Constraints and Objectives for the SMEF

The goal of the experimentations reported in this paper is to develop a *Specialized Model-based Engineering Framework* (SMEF) that supports the *Secure Design Engineering Process* (SDEP) succinctly described above. Contrary to the majority of model-based engineering processes, two different stakeholders are actively interacting with the model. One of them will always be the same (the CA) while the other can be many different industrial enterprises acting as PCs. Those enterprises may (or may not) have their own model-based engineering processes. It would be nearly impossible to drastically change at once some of the processes of the CA, and it would be difficult to impose a replacement of engineering processes to enterprises working as PC (at least not for a reasonable cost). Therefore, *the proposed SMEF must cooperate as smoothly as possible with existing document-based processes used by the CA and with generic industrial engineering processes.*

Modeling is quite often a daunting task. Users tend to get overwhelmed by the modeling possibilities and get lost between all the potential "next steps" (adding information in depth or breadth, and under which format) involved in modeling a realistic product behavior and structure. Modeling requires a strict discipline. In order to facilitate the acceptance of the SMEF, *the framework should come with a precise methodology explicitly and clearly identifying what to model, when, and using which modeling elements.*

Related to the number and nature of active participants (main stakeholders and teams inside them) as well as "sets of information" (documents), during an instance of the SDEP, pieces of information are regularly exchanged: between the two main stakeholders (CA and PC); as well as between sets of information (documents) and their associated teams inside the stakeholders (Fig. 2). Due to security reasons, those pieces of information can not be shared on a common network resource (access to the Internet is restricted and no common network smoothly spans both stakeholders). At any given time, the CA and PC work on different "subset copies" of the same "set of information" (with organizationally limited overlap on information updates). Regularly, information is exchanged to asynchronously update the different copies. In order to be applicable in this setting, *the SMEF should facilitate asynchronous updates of model copies by offline channels.* Additionally, inside each of the two main stakeholders, different participants may be concurrently working on the same "subset of information". Hence *the SMEF should also support the concurrent interaction with a model and version control of models.*

In order to facilitate the acceptation of the SMEF, the SMEF should comply with some of the practices of the document-based SDEP. *The SMEF should for example allow the automatic generation of the main parts of the CdS and DEC documents from the model.* As the security analysis in the document-based SDEP relies on attack trees [17, 23], *the SMEF should support security analyses based on attack trees* and integrate them in the model.

Finally, one of the main motivations to experiment with a SMEF for the SDEP is to enhance the overall quality of the different "information subset copies". Therefore, *the SMEF should help ensure and verify the completeness and correction of the model.*

***Summary of the SMEF's high level requirements:***

- ($HLR_1$) Must be based on a precisely described methodology guiding the framework users;
- ($HLR_2$) Must be applicable with the vast majority of enterprises working on the development of secure IT products;
- ($HLR_3$) Must be compatible with the document-based secure design engineering process (SDEP) highlighted in Fig. 1 involving hardware and software design;
- ($HLR_4$) Must support collaborative work by facilitating: the offline exchange of pieces of data between different copies of the same model, and concurrent modification of a model;
- ($HLR_5$) Must support the realization of security analyses based on attack trees;
- ($HLR_6$) Must allow the nearly complete generation of CdS and DEC;
- ($HLR_7$) Must reduce the incompleteness, ambiguities and inconsistencies in the information exchanged between the contracting authority and prime contractor.

Overall, it must be a model-based version of a document-based process. The interactions between the final users and the models must then closely reflect the content of the current documents. As a consequence, similar approaches, as SysML-Sec [2], can not be directly applied.

## 3 Realization / Work Done

Some of the experimentations work has been done in part by contracts with companies. At the time of writing, all contracted work is not finalized yet and the modeling tool of the framework is thus still a property of one of the contracted companies. However, important cooperation with and supervision of the companies' work, as well as intensive verification operations, have allowed us to extract a first set of results from those experimentations. Those findings are exposed and discussed in the following section (sect. 4), while the current section describes succinctly the work achieved.

Early in the experimentation, it has been decided that the SMEF would be built on top of SysML, and thus that the metamodel used would be a SysML profile. This choice has been made with the aim to satisfy the high level requirement $HLR_2$ and comply with the fact that SysML version 1.4 is an agreed NATO standard for system modeling [9]. By choosing a well-known and widely available standard, the goal is to facilitate

the integration of the SMEF with current and future model-based engineering processes potentially used by enterprises working with contracting authorities on the development of secure IT products and to facilitate the comprehension of the information carried by the SMEF models.

***Definition of a precise and specialized methodology.*** The first work achieved in this experimentation is the clear definition of a specialized methodology incorporated to the SMEF. This methodology defines:

- the process to follow, identifying the different tasks to achieve, their precedence, the information exchanged between those tasks, the information to define in each task, and the documents or artifacts produced by each task;
- the meta-model used, defined as a SysML profile, to model those information pieces;
- and the "encoding" of information to be used, i.e. how to model an information among the various ways an information can sometimes be modeled in SysML.

The overall process is not divided into tasks performed by the contracting authority (CA) and tasks performed by the contractors. The process is defined along the different modeling steps. During the execution of this process, the model is created mainly (but not always) by the contractor and the current model is regularly sent to the contracting authority for analysis. Figure 3 provides an overview of the modeling process. The SMEF process is divided between tasks performed by security analysis teams and tasks performed by architecture design teams.

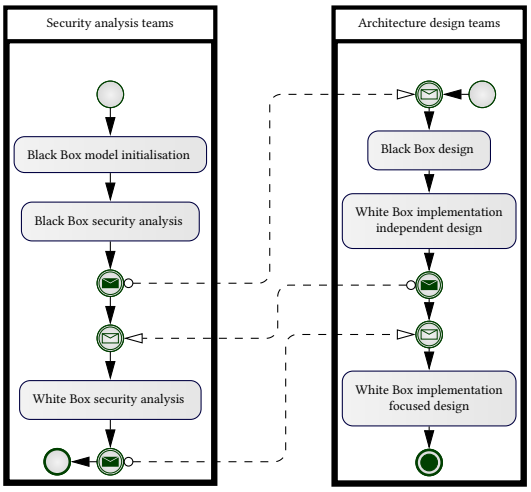



**Figure 3.** Overview of the SMEF process (BPMN)

The first major step consists in the initialization of the black box specification of the future product and its security analysis. This task is usually initialized by the CA's security analysis team and finalized by the contractor's security analysis team. This step is refined in a few different tasks. *Hypotheses*, *implementation constraints* and *security policies* (as defined in the common criteria [25]) are first created from scratch or instantiated into the model from libraries (a common criteria library is provided with the framework). Then, the *services* provided by the product and known *assets* manipulated by the product are modeled. At this stage, *services* are modeled as `UseCase` objects extended with a specific stereotype inheriting from the `Security_Operation` stereotype and storing additional information relevant for the SDEP. *Assets* are modeled as `DataType` objects extended with a specific stereotype inheriting from a generic `Asset` stereotype to store additional information related to security properties. The different types of human *actors* interacting with the future product (administrators, regular users, remote users, auditors, ...) are modeled as `Actor` objects extended with an `Agent` stereotype. Once *actors* are registered into the model, the hierarchy of *actors* is modeled in a specialization of UseCase diagrams called Role diagrams. And the accessibility of *services* to *actors* is modeled in other specialized UseCase diagrams, called Service/Function diagrams, using association relations. *Security functions* (providing security related features) are then modeled as `UseCase` objects extended with a specific stereotype inheriting from the `Security_Operation` stereotype; and, using the same type of diagrams (Service/Function diagrams), the *security functions* used by the *services* are defined using "include" relations. Then the *environment entities* of the product are modeled as blocks in a specific Block Definition diagram (BDD); and, in a specific Internal Block diagram (IBD), the *interfaces* of the future product are modeled as stereotyped `FlowPort` objects providing additional information such as the *services* available on those *interfaces* and the *assets* transiting on those *interfaces*. In this same IBD, the *connections* between the future product and its environment entities are modeled as stereotyped `Connector` objects which provide information concerning the *assets* exchanged on those links. Finally, the future product *life cycle* is modeled in a specific StateMachine diagram using stereotyped `State` objects.

Once all those elements are modeled, an attack tree is automatically initialized by combining information on *assets*, *services*, *security functions*, *life cycle states* and *interfaces*. This attack tree is then analyzed and completed to expose all *threats* and cover each of them with an *environment objective* or *technical objective*. *Environment objectives* are then translated into *Security Assurance Requirements* (SAR); and *technical objectives* are translated to *Security Functional Requirements* (SFR) assigned to *security functions*.

At this stage a first version of the CdS document can be generated and the architecture design teams can start their work. The first step is to model the external communication *protocols* as stereotyped `Package` objects containing Sequence diagrams defining the message exchange process. *Protocols* are modeled as `packages` in order to facilitate the

reuse of their definitions by relying on import/export features of modeling tools. Once *protocols* are modeled, the *connections* previously modeled are completed to specify which *protocols* are used.

Then the functional decomposition (behavior) of *services* and *security functions* are modeled using Activity diagrams. When defining the behavior of a *service* or *security function*, an `Activity` object is associated to the `Security_Operation` stereotyped `UseCase` object modeling this *service* or *security function*. This `Activity` object is also stereotyped `Security_Operation` with the same field values. It is this `Activity` object for which an Activity diagram is constructed. This activity diagram is mainly composed of `CallBehavior` objects related to a *service* or *security function* and `Opaque Action` objects.

Before being able to generate a first version of the DEC document, it remains to create coverage links in the model between *SFR* and *security function* behaviors in such a way that, for every *security function*, every *SFR* assigned to it is covered by at least one `CallBehavior` or `Opaque Action` of its associated Activity diagram.

Then the architecture design teams can move to the modeling of the white box design. The first task to accomplish is to model "views" on the overall behavior of the future product depending on: power sources available; authentication state; and functional state. This is done by creating specialized StateMachine diagrams whose `State` objects are stereotyped `Power_State`, `Authentication_state` or `Functional_State`. Those stereotypes contain a field allowing to enumerate the *services* potentially available in those states. The next task is to describe the implementation independent architecture (similar to a PIM) using BDD and IBD whose blocks are stereotyped to indicate that those blocks are implementation independent and may receive additional stereotypes to indicate that they belong to specific "domains", such as the administration domain. *Services* and *security functions* are projected on those blocks as `Operation` objects stereotyped again with the `Security_Operation` stereotype. Once again, the values of the fields of those stereotyped objects must match the values of the stereotyped `UseCase` and `Activity` that represent the same *service* or *security function*. Stereotyped `FlowPort` objects are also added to the implementation independent blocks to represent *functional interfaces*. Those *interfaces* provide information such as: the *assets* transiting through them, the *functions* that can be called, and their accessibility to different *actors*.

Then the new *security functions*, *assets* and *interfaces* are exported and sent to the security analysis teams. Those elements are imported and the security analysis is updated to take into account those new elements. This may give rise to new elements, including new SFR. Elements relevant to the design teams are once again exported and sent to the design teams which import them. The implementation independent white box model (PIM equivalent) is updated if need be, and the design teams move to implementation focused white box model (equivalent to a PSM) executing similar tasks as for the implementation independent white box model.

***A specific model hierarchy.*** The security analysis and architecture design teams have different views on the model which can be roughly divided in three models: a Black Box Model (BBM), Implementation Independent white box Model (IIM) and Implementation Focused white box Model (IFM). The initial model of the analysis contains only the BBM, which is exported to the design team after the security analysis (in addition, there is an attack tree model which is kept separate and loosely linked to the BBM, and is never exported to the design teams). The model of the design team contains all 3 sub-models and its BBM is initialized from the one imported from the analysis team. The design team exports to the analysis team its BBM and some elements from its IIM. After updating the security analysis, the analysis team exports its new BBM and IIM.

In order to facilitate those export/import operations between the analysis and design teams, the overall model is devided into (Papyrus) sub-models: one for the BBM, one for the IIM and one for the IFM. In addition some elements, such as the future product top architecture block, is duplicated in each sub-model; while other elements, such as assets or actors, must be created only in the BBM and referenced in other sub-models. This is a compromise that allows BDD and IBD in the sub-models to be self-contained (no blocks are reference to blocks in other sub-models) while avoiding excessive duplication.

Finally, in order to facilitate the navigation in a model that may become quite big, there is a default model hierarchy to follow. Elements such as actors, assets, services and security functions have to be created in specific packages, mainly in the BBM. Other packages are dedicated to the description of specific aspects such as the product life cycle, the product environment, and external communications.

***Development of a dedicated specialized modeling tool.*** In addition to the definition of a dedicated methodology, one important part of the experimentation is the development of a specialized modeling tool that supports and facilitates the use of the SMEF.

This tool is heavily based on Eclipse Neon 3 and Papyrus 2.0.3 (https://www.eclipse.org/papyrus/). In addition, the Xmind tool (https://www.xmind.net) has also been integrated in order to manipulate the attack trees inside the modeling tool. And ReqCycle (https://www.polarsys.org/projects/polarsys.reqcycle) has been integrated in order to handle the high level security analysis requirements and notions (including the CC ones such as SFR, SAR, and technical objectives). All those tools have been integrated in such a way that traceability links can be created between elements manipulated by those tools (attack trees, requirements, and product model). The modeling tool also integrates

MARTE's library (https://www.omg.org/omgmarte) and is able to generate documents from the models using GenDoc (https://www.eclipse.org/gendoc).

The user interface of the modeling tool has been optimized for the use of the SMEF's SysML profile: dedicated (contextual) SMEF sub-menus; "new" dedicated diagram types inheriting from SysML/UML diagrams; and optimization of diagram palettes. For example, the contextual menu of the "Roles" package proposes the creation of role diagrams (inheriting from UseCase diagrams) whose palette allows only the creation of "Agent" objects (stereotyped actors) and generalization links.

The SMEF methodology (based on a SysML profile) requires to create different (stereotyped) SysML objects for a single "semantic" object (or notion). For example, in the SysML model, a *service* (or *security function*) is represented as a `UseCase` (for association in UseCase diagrams with actors and included security functions), an `Activity` (for association with other model elements, such as behavioral description or calls in Activity diagrams), and an `Operation` (for projection into blocks implementing this service). In order to facilitate the preservation of the consistency between those model artifacts representing the same "semantic" object, some automatic behaviors have been implemented in the modeling tool. For example, a drag'n drop of a `Security_Operation` stereotyped `UseCase` object into a block automatically creates into that block a `Security_Operation` stereotyped `Operation` with the same field values as the `UseCase` and associated with an `Activity` itself associated with the originating `UseCase`.

The SMEF methodology also defines quality related rules (e.g., every interface is typed, or every function is projected on the architecture) and security related rules (e.g., sensitive data is manipulated in accordance with its sensitivity level). The modeling tool also provides features that help users to comply with those rules at conception time (coloration depending on projection status or sensitivity level of data manipulated) and at verification time (OCL-based [20] and Java-based verification rules).

## 4 Observations

Interesting observations started to emerge from the experimentation as early as during the design phase of the methodology and modeling tool, and continued to emerge in all the experimentation stages including a four months long contractual verification operation involving the design of a somewhat realistic product.

The current section reports on the main findings, some of which are more widely discussed in sect. 5. It first describes usability-related observation, some of which are related to the implementation-related observation or to the design-related observation.

***Usability-related observations.*** The first point to emphasize is the importance of the clear and explicit definition of an associated methodology. A first benefit relates to the support its definition provides to the design of the framework. The precise definition of the methodology associated with the framework requires the clear definition of the information that has to be modeled and of the information that is exchanged between the different participants involved in applying the framework. Knowing precisely what information needs to be modeled facilitates the creation of a clear domain model dedicated to the precise process supported, which gives rise to a meaningful and dedicated meta-model (or profile). Whereas knowing in advance which subsets of this information are exchanged helps structure the meta-model in a way that facilitates (in the limits of what is possible) the required export and import of model fragments. A second benefit to the clear and explicit definition of an associated methodology is that it helps users apply more easily the SMEF. The first part of the SMEF user manual describes the methodology. It describes the successive steps to follow to execute the SMEF, what has to be done in every step, what information has to be modeled. The first part of the manual does not state how to model those information pieces; with regard to the how, it only references sections in the second part of the manual which explain how to use the modeling tool to do those modeling operations. The first part of the user manual, which we found to be the most useful, focuses on what to do in the current stage, and not how to do it. Clearly knowing what to do and when tremendously help users apply the framework. Users do not get lost in front of all the possibilities offered by the modeling tool.

With a similar consequence for users, modeling tool customization removes unnecessary information and actionable items away from the user, leaving only information and operations that are relevant to execute the framework at this stage. By offering a SMEF-specific contextual menu for nearly every object, the modeling tool simplifies questions that users can ask themselves, such as: what diagram to create to define an object, what class or relation to use to express a specific information, or what type of object (child) to create in a given package. As with an explicit methodology, a customized modeling tool makes it easier for users to use the framework and facilitates its adoption.

The current implementation of the *Secure Design Engineering Process* (SDEP) relies on word-processing programs to produce the two main documents shared between the contracting authority and prime contractor (CdS and DEC). Maintaining the accuracy and consistency of those documents through out the SDEP using word-processing programs is a difficult task. As the content and structure that those documents have to respect are clearly specified, it has been possible to ensure that the majority of information they contain is included in the models and to generate decent quality almost complete CdS and DEC from the model.

This would allow to move from a document-based SDEP to a model-based SDEP by going through a transition period where both processes are used and the documents are produced from a model. It is our belief that it can be useful, when going from a document-based process to a model-based process, to ensure that the majority of the documents used can be generated from the models.

This experimentation has also given rise to less positive observations. As stated in sect. 3, the SMEF's model follows a specific hierarchy facilitating navigation in the model and export of sub-models. However, it has been found difficult to respect this hierarchy during the whole modeling process. By defaults in Papyrus, objects in a diagram are created under the diagram's root object. As the model grows in complexity and objects are created driven by the diagrams need, the model hierarchy tends to be less and less respected. It has been found that, in order to respect the hierarchy, all objects have to be first created in the model explorer and then incorporated in the diagrams. However, this approach requires a strict discipline that is hard to follow. This may potentially be solved by even more modeling tool customization in order to create model objects in specific places respecting the model hierarchy. However it is not obvious if it is possible to define implementable rules that are not too restrictive.

Similarly, preserving the model consistency is difficult. In part because the meta-model is a SysML profile (which is discussed further below), many different objects and field values have consistency relations (e.g., they should have the same values, or should relate to the same objects or a subset of objects). Preserving the model consistency manually is doable, but really difficult. Some of the verification rules help verify model consistency, but sometimes their implementation is itself based on the preservation of the consistency of some relations or values. It should be possible to facilitate the preservation of the model consistency by increasing the modeling tool customization, but this would reduce flexibility and increase the implementation complexity.

On a related and final usability point, the current implementation of the modeling tool has some stability issues that are related to implementation difficulties. The first one relates to the customizability of Papyrus/Eclipse which appears to not be as easy as it may seem (everything being relative). While some lightweight customizations seem easy to implement. The contracted company had difficulties to implement some of the more "invasive" modifications. For example, the company has been unable to deactivate the default verification rules related to UML profiles and MARTE. Another implementation-related issue is the relatively low stability and robustness of the tool. Seemingly minor exceptions (mainly NullPointer) pop up from time to time and seem to gradually degrade the underlying model.

***Design-related observations.*** The first design-related observation is not really new and is a point addressed in many information-related models. However, it is a point that has been overlooked while designing the SMEF meta-model and is worth mentioning again. For information related modeling, it is important to distinguish between information (the content) and data (the container), and to capture the interesting relations between those elements. The current SMEF meta-model does not do this distinction and does not capture the interesting relations between and among information pieces and data. There is only one class (with inheriting subclasses for types of data such as cryptographic keys, logs, …) with a classification field providing information on the impact on the loss of this data. Therefore an unencrypted information and the same information encrypted are represented by two different objects having no direct relations. When designing an information related meta-model it is important to distinguish between data and information, and to make sure the meta-model embeds information relations that are relevant for the models.

This experimentation also emphasized the difficulty to embed a process-specific model (and its meta-model) into SysML. Under this perspective, SysML does not closely follows the main philosophy of the family of Model-View-★ design patterns [24]. There is not a unique model where process-specific (or domain) "entities" are instantiated only once, and multiple partial views (diagrams) on that model. In fact, it is as if there is one model per diagram (view) type. Those models partially overlap and are partially coupled by relations; but there aggregation does not form a unique model where domain "entities" are instantiated only once. As just exposed, many SMEF-specific entities had to be represented by different SysML objects in different diagrams. For example, *services* or *security functions* are represented by many different objects among which:

- a `UseCase` instance in order to capture accessibility by users and inclusion relations in UseCase diagrams;
- an `Activity` instance in order to describe its behavior in an Activity diagram and for linking with other model elements;
- some `CallBehavior` instances to reflect execution of the corresponding *function* in another *service* or *function* behavioral description;
- an `Operation` instance in order to register which architecture block implements this *function*;
- some `Behavior Execution Specification` instances to represent its execution in Sequence diagrams;

The differences between all those SysML classes are irrelevant for the SMEF. All those occurrences could be represented by a single element in the SMEF model. However, in order to use SysML and its diagrams, it is necessary to use many different model elements to represent the same "domain" object. Finding which SysML classes to use to represent a "domain" object, and which relations to use to link all its SysML model elements, has been not so obvious. Some

of the SysML objects that have to be created do not even explicitly appear in the diagrams.

The multiplication of modeling elements to represent the same "domain" object creates model consistency issues. It is desirable that a *service* or *security function* be identified by a unique name. Should all the `UseCase`, `Activity`, `CallBehavior`, `Operation`, `Behavior Execution Specification` instances have consistent values for their name field? Or should only the `UseCase` name field be relied on and all the other name fields be ignored? But then, would not that be error prone for the user? Should modeling tool automations ensure name consistency for all the instances? Is it easy to implement correctly? If a `UseCase` already linked to an `Activity` is linked to a different `Activity`, which ones of the 3 instances need to have their name changed? Without automations however, keeping the model consistent requires a lot of work from the users. By analogy with the Model-View-ViewModel design pattern (MVVM) [19], the SysML diagrams are the views, the SysML model is the agglomeration of the ViewModels, but the model itself (where domain entities are instantiated only once) is missing. Potentially, a better design pattern for the SMEF would be to follow the MVVM pattern where a Domain-Specific Modeling Language would be used for the model, and SysML would be used only for ViewModels and Views. This would however give rise to other difficulties linked to the bidirectional nature [1] of transformations between Models and ViewModels.

## 5 Discussion

Two observations deserve a deeper discussion. The first one concerns the distinction between model (abstract) syntax and model semantics, and the relation to verification operations. The second observation concerns the intrinsic reason why it is difficult to embed specific ad'hoc models into SysML. The statements expressed in this section *may* be controversial for some. This section expresses the authors' personal thoughts triggered by the experimentation work. They are not stated as conclusions, but as an opening discussion.

### 5.1 Distinguishing Model Syntax and Semantics

One of the lessons learned from this experimentation is that it is important, while designing process-specific "modeling support" (the meta-model, or modeling language, that will be used), to remember that a model has a syntax and a semantics, and that they are distinct. Syntax entities are *interpreted* to give rise to semantics entities. Users interact trough "objects" belonging to the syntax domain to express "information" belonging to the semantics domain. Verification rules are applied on syntax entities to check that semantics entities have the right properties. Standard meta-model definition defines model syntax, but semantics has not to be forgotten. Syntax, semantics and their relation can, and should, be optimized to facilitate model manipulation by users and model verification.

For example, in SMEF models, *functions* are decomposed in (sub-)*functions*, architectural *components* are decomposed in (sub-)*components*, and every *function* is projected on a *component* (identifying which component implement which functions). A semantically complete and correct SMEF model: a) has all its functions projected on a component; b) and a sub-function has to be projected on the same component (or a sub-component) as its parent function. Semantically, the projection relation between *functions* and *components* is a many-to-one relation ($\star - 1$). There are various ways to syntactically represent this semantic relation, among which the two following ones.

**direct representation** The relation can be directly represented in the model syntax by a many-to-one association ($\star - 1$).

**indirect representation** It can also be represented by an "optional" association ($\star - 0..1$) with the semantic interpretation that, if a *function* is not syntactically projected on a component, then it inherits from its parent semantic projection or, if it does not have a parent, it is projected on the product top architectural component.

The direct representation involves more associations, that have to be created and maintained. If the projection of a function is modified then the projection of all its sub-functions also have to be modified. The direct representation implies more work for the user and/or the modeling tool automations. From a verification point of view, using the indirect representation, the first property (a) is always true, by virtue of the semantic interpretation, and the second property (b) is also always true for functions that are not syntactically projected. For this particular case, the indirect representation increases both usability and efficiency.

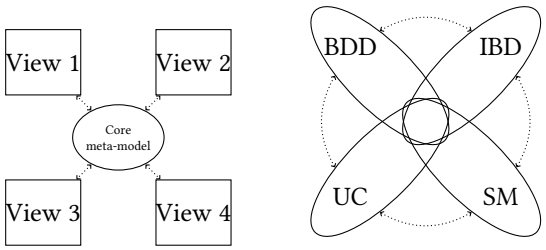
### 5.2 SysML is a Meta-model of "Views"

SysML, as a recognized general purpose system modeling language standard, is a good mean to communicate on different aspects of an IT security-related product engineering design. However, by observing drawings of SysML meta-model graphs, one can observe that the classes associated to the model elements used by different diagram types form distinct "diagram cliques" sharing little to no elements.

This sighting is reinforced by the observations made when designing the SMEF's SysML profile and the way to use it (Sect. 4). Many elements in the SMEF's semantic domain are represented by instances of multiple SysML classes: one main instance per diagram type. For example, every *security function* is represented by many different SysML objects, including: one `UseCase` instance for representation in UseCase diagrams (the single instance can be reused in every UseCase diagram but not directly in any other diagram

type); one `Activity` instance to be used as representation of any Activity diagram root or linked to any call; one `Operation` instance to represent the *security function* in any Block Definition diagram; and, a few `Behavior Execution Specification` instances for representation in Sequence diagrams.

This seems to suggest that a SysML (or UML) model is not a model of a system with diagrams being views on that model. Rather, it is an agglomeration of graphical diagram models. SysML is an agglomeration of diagram meta-models (Fig. 4b), rather than a single core meta-model with the ability to extract (and interact through) "views" which may have their own meta-models (Fig. 4a).




**(a)** Core meta-model + views **(b)** Diagram meta-models agglomeration

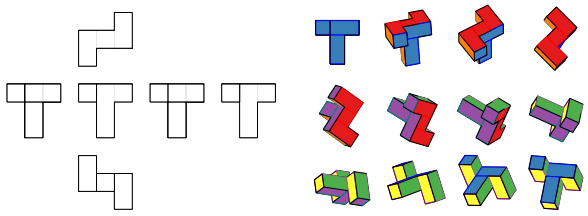
⟷ Direct consistency constraints between element instances

**Figure 4.** Metamodel structures

This share similarities with technical engineering drawings [6, 10, 15, 28]. Historically, three dimensional objects were mainly modeled, represented and communicated upon using orthographic, a.k.a. multiview, projections (first-angle or third-angle), similar to the one of Fig. 5a. In those drawings, there is no direct syntactic model of the object per se; rather there are some syntactic models of views on that object. Those syntactic models can be interpreted to get semantic models of those views; and, by combination of those semantic models and intellectual deduction, a reader can conceptualize a semantic model of the object itself. However, nowadays with the advent of 3D CAD models, it is possible [1] to directly construct syntactic (numerical) models of the object itself; and the multiview projections (Fig. 5a), which are still useful for communication, are automatically extracted from the object model (as for the axonometric projections of Fig. 5b).

The current state of UML/SysML may be due to historical reasons. Indeed, the majority of UML diagrams syntax and semantics were first created in different contexts (a variant of nearly every UML diagram types existed prior UML) and they were all put together in a single standard, reusing their syntax and semantics elements, and creating links between

[1] 2D CAD drawings are also a pertinent way to model 3D objects



**(a)** Orthographic **(b)** Axonometric

**Figure 5.** Projections of the same 3D object

them. It is not clear to the author if the fact that UML/SysML is an agglomeration of views meta-models is only due to historical reasons, or if there is a *deeper intrinsic reason*. It may be the case that, for general purpose system modeling languages, the only thing that can be generically defined are the means to interact with and communicate on the model, because an efficient core meta-model is too tightly linked to the specificities of the fine purpose or application domain of a model. Is it possible to construct a core meta-model for UML/SysML: small enough to be usable and not too complex; and expressive enough to serve as a general purpose system meta-model? Is it possible to do it for a general purpose system modeling language, or is it only doable for "purpose" specific modeling languages (i.e. languages even more specialized than domain-specific ones)?

## 6 Conclusion

This paper reports on experiments to develop a specialized model-based engineering framework (SMEF) for a design engineering process of security-related IT products (SDEP). The first step of this experimentation has been the definition of a detailed and explicit methodology associated to the SMEF. This definition work helped precisely define, as a SysML profile, the meta-model used in the SMEF. Then a specialized modeling tool supporting use of the SMEF as been developed by customization and integration of different tools such as Xmind, ReqCycle and Eclipse/Papyrus.

Different observations have been made during the experimentation. Some of them relate to the usability of the resulting framework and emphasize the importance of an explicit and clear methodology. Such a methodology helps user navigate the framework process. It also emphasized the difficulty to project (or embed) an ad hoc process specific model into a general purpose modeling language such as SysML. For many domain concepts, every single instance has to be represented by different SysML objects depending on the views in which it is used. Which gives rise to a complex meta-model having many consistency rules to respect between the different elements of the model. Some of those observations open up discussions on: the importance of the distinction between syntax and semantics of models for the

design of the meta-model and verification rules; and the perception of the UML/SysML meta-model as an agglomeration of view meta-models.

Overall, the SMEF developed is usable "by fragments" to model some aspects of the product. However, when used for big models covering all aspects of interest for the SDEP, consistency issues start to arise and are difficult to keep up with. On top of that, some stability issues of the modeling tool decrease its usability for big models.

It is not clear if a satisfying result could be achieved in a tractable way using SysML. Its "agglomerate" nature makes it difficult to use as a "core meta-model". However, SysML models are useful for communication. A more promising approach may be to design a process specific meta-model to construct the internal syntactic model of the product itself, and translate it to "view" models for interaction (in a way similar to what is done for 3D CAD models), and to SysML models for communication.

## Acknowledgments

The author wishes to thank the reviewers whose comments helped improve this paper. Some of the insightful references provided [3, 4, 8, 11, 13, 18] could not be integrated due to lack of time, but will be in an extended version.

## References

[1] F. Abou-Saleh, J. Cheney, J. Gibbons, J. McKinna, and P. Stevens. 2018. Introduction to Bidirectional Transformations. In *Bidirectional Transformations*. Lecture Notes in Computer Science (LNCS), Vol. 9715. Springer, 1–28. https://doi.org/10.1007/978-3-319-79108-1_1

[2] L. Apvrille and Y. Roudier. 2013. SysML-Sec: A SysML Environment for the Design and Development of Secure Embedded Systems. In *Proc. Conf. System Engineering*.

[3] C. Atkinson, D. Stoll, and P. Bostan. 2010. Orthographic Software Modeling: A Practical Approach to View-Based Development. In *Proc. Int. Conf. Evaluation of Novel Approaches to Software Engineering (Communications in Computer and Information Science)*, Vol. 69. Springer Berlin Heidelberg, 206–219.

[4] C. Atkinson, C. Tunjic, and T. Möller. 2015. Fundamental Realization Strategies for Multi-view Specification Environments. In *Proc. Int. Enterprise Distributed Object Computing Conf.* IEEE Computer Society, 40–49. https://doi.org/10.1109/EDOC.2015.17

[5] D. M. Berry, E. Kamsties, and M. M. Krieger. 2003. From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity – a Handbook. Online. http://se.uwaterloo.ca/~dberry/handbook/ambiguityHandbook.pdf

[6] G. R. Bertoline. 2008. *Introduction to Graphics Communications for Engineers* (4th edition ed.). McGraw-Hill Education.

[7] J. Bézivin, S. Gérard, P.-A. Muller, and L. Rioux. 2003. MDA components: Challenges and Opportunities. In *Proc. Work. Metamodelling for MDA*. 23–41. https://hal.archives-ouvertes.fr/hal-00448057

[8] H. Bruneliere, E. Burger, J. Cabot, and M. Wimmer. 2018. A Feature-based Survey of Model View Approaches. In *Proc. Int. Conf. Model Driven Engineering Languages and Systems*. ACM, 211–211. https://doi.org/10.1145/3239372.3242895

[9] C3B Interoperability Profiles Capability Team. 2019. *NATO Interoperability Standards and Profiles*. Allied Data Publication 34(L). NATO C3B. Version 12.

[10] I. Carlbom and J. Paciorek. 1978. Planar Geometric Projections and Viewing Transformations. *ACM Comput. Surv.* 10, 4 (Dec. 1978), 465–502. https://doi.org/10.1145/356744.356750

[11] J. de Lara, E. Guerra, J. Kienzle, and Y. Hattab. 2018. Facet-oriented Modelling: Open Objects for Model-driven Engineering. In *Proc. Int. Conf. Software Language Engineering*. ACM, 147–159. https://doi.org/10.1145/3276604.3276610

[12] S. Gérard, C. Dumoulin, P. Tessier, and B. Selic. 2010. Papyrus: A UML2 Tool for Domain-specific Language Modeling. In *Proc. Model-based Engineering of Embedded Real-time Systems*. Lecture Notes in Computer Science (LNCS), Vol. 6100. Springer Berlin Heidelberg, 361–368. https://doi.org/10.1007/978-3-642-16277-0_19

[13] E. Guerra. 2018. On the Quest for Flexible Modelling.

[14] W. H. Higaki and Y. Higaki. 2010. *Successful Common Criteria Evaluations: A Practical Guide for Vendors*. Createspace.

[15] ISO. 1996. *Technical drawings — Projection methods — Part 2: Orthographic representations*. Standard ISO 5456-2:1996(en). International Organization for Standardization.

[16] E. Kamsties. 2005. Understanding Ambiguity in Requirements Engineering. In *Engineering and Managing Software Requirements*. Springer, 245–266.

[17] P. A. Khand. 2009. System level security modeling using attack trees. In *Proc. Conf. Computer, Control and Communication*. 1–6. https://doi.org/10.1109/IC4.2009.4909245

[18] L. Lucio, S. bin Abid, S. Rahman, V. Aravantinos, R. Kuestner, and E. Harwardt. 2017. Process-Aware Model-driven Development Environments. In *Proc. Int. Conf. Model Driven Engineering Languages and Systems*.

[19] Microsoft. 2012. The MVVM Pattern. https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)

[20] OMG. 2012. *Object Constraint Language (OCL)*. Standard ISO/IEC 19507:2012(E). International Organization for Standardization. http://www.omg.org/spec/OCL/ISO/19507 Version 2.3.1.

[21] OMG. 2012. *OMG Systems Modeling Language (OMG SysML$^{TM}$)*. Standard. Object Management Group (OMG). http://www.omg.org/spec/SysML/1.3/ Version 1.3.

[22] OMG. 2014. *MDA Guide revision 2.0*. OMG Document ormsc/2014-06-01. Object Management Group (OMG).

[23] B. Schneier. 1999. Attack Trees. *Dr Dobb's Journal* 24, 12 (Dec. 1999). https://www.schneier.com/academic/archives/1999/12/attack_trees.html

[24] A. Syromiatnikov and D. Weyns. 2014. A Journey through the Land of Model-View-Design Patterns. In *Proc. Conf. Software Architecture*. 21–30. https://doi.org/10.1109/WICSA.2014.13

[25] The Common Criteria Recognition Agreement Members. 2006. Common Criteria for Information Technology Security Evaluation. http://www.commoncriteriaportal.org/

[26] The Eclipse Foundation. 2019. Eclipse: The Platform for Open Innovation and Collaboration. https://www.eclipse.org/.

[27] The Eclipse Foundation. 2019. Papyrus: Modeling environment. https://www.eclipse.org/papyrus/.

[28] Musashino Art University. 2018. Art & Design Glossary: Third Angle Projection. http://art-design-glossary.musabi.ac.jp/third-angle-projection/.