Validation of a security metamodel for development of cloud applications

Marcos Arjona[1], Carolina Dania[2], Marina Egea[3], Antonio Maña[1]

[1] Universidad de Málaga, Spain
[2] IMDEA Software Institute, Madrid, Spain
[3] Atos, Madrid, Spain

marcos@lcc.uma.es, carolina.dania@imdea.org,
marina.egea@atos.net, amg@lcc.uma.es

Abstract. Development of secure cloud applications requires a supportive approach that should also enable software assessment and certification by different mechanisms. These can assure by independent means that the required security is present. In this paper we present a Core Security Metamodel (CSM) that is the director of a security engineering process that also addresses security certification for cloud applications. To drive these activities with enough precision, the CSM is constrained with OCL rules that control the creation of instances of the metamodel. Due to their relevance for the security engineering process, we decided to formally check their consistency leveraging on our previous mapping from OCL to First Order Logic. We found that CVC4 returned *sat* in less than 30 seconds when we run it in finite model finder mode. Also, it automatically provided a valid CSM structural instance. Instances so obtained with CVC4 can be tuned to serve as input of the engineering process of secure cloud applications. Their automatic generation reduces the time and effort spent in the engineering process, reinforcing its supportive and practical side.

1 Introduction

Development of secure applications is a challenging task due to the evolvable risks that threaten any system under design. Even worse nowadays, the exposure of systems to cloud environments claims for a stronger development approach able to support a large number of complex security requirements and interplay in the creation of cloud applications. Most of the proposed approaches agree in the necessity to sit a solid and affordable engineering process that can prevent, from design time, non-secure states due to wrong security mechanisms used as a late solution [18]. In line with this approach, our work stems in the definition and evaluation of a security engineering process [6] for the CUMULUS [1] and the PARIS [4] EU projects. Our work proposes a complete Model Based System Engineering (MBSE) methodology to address the different stages involved in the development of secure and privacy preserving applications. It includes early stages of the architectural design that identify the security requirements. Also, it covers subsequent stages in which solutions are manipulated and system model transformations progressively applied up to the inclusion of all the security mechanisms that fulfill those security requirements.

In this paper, we are focused at the first stage of the work flow: the Core Security Metamodel (CSM), designed to gather and represent the security knowledge. The CSM and the OCL validation rules imposed on it establish a language that supports, validates and drives instance creation and subsequent steps of the engineering process. Due to their relevance for the security engineering process, we decided to formally analyze them. Thus, we mapped them to First Order Logic following our previous work [13,14,21] and then used off-the-shelf tools to run the analysis. We run Z3 [15] and CVC4 [7] as SMT solvers in the first place but they did not help us with the consistency checking. Then, we employed CVC4 as a finite model finder, which returned *sat* in less than 30 seconds and automatically provided a CSM valid structural instance. Instances so obtained are manually enhanced with security knowledge later on to serve as inputs of the engineering process of secure cloud applications. Still, their generation can reduce the time and effort, reinforcing its supportive and practical side.

Organization. In section 2 we outline related work. In section 3 we introduce the CSM, its OCL constraints and its intended use. In section 4 we summarize our previous mapping from OCL to first order logic and illustrate how CSM rules are mapped. In section 5 we report on the CVC4-based validation and instance generation. In section 6 we illustrate how instances can be enhanced to drive subsequent steps of the engineering process. Finally, in Section 7 we present conclusions and directions for future work.

2 Related work

In the software engineering arena there are a number of ready-to-use tools supporting OCL. Possibly the best starting point to get introduced to a variety of them is the OCL Portal.[4] Most of the tools that it contains ($\sim$ 11) are OCL parsers or evaluators. Also, there are 3 static verification tools, one code generator and one OCL transformation tool. For this related work, we focus in the OCL automatic verification or validation tools that could help us as alternative or complementary formal analysis means to the ones that we have already applied to CSM helped by Z3 [15] and CVC4 [7]. [5]

Recently published, the systematic review [17] deeply reports on 18 research lines on static verification of UML-like structural diagrams. Taking these research results as the starting point, we decided to focus here only on those for which the tool associated is ready to download or use from a website and supports automatic analysis for a significant subset of OCL. We consider these criteria as essential criteria for analysis tools to be actually of use in real development processes.

Alternative tools to CVC4 for our goal could be, in principle, the ones reported next. UMLtoCSP [12] and EMFtoCSP [16] both provide bounded automatic verification of UML (resp. EMF) models annotated with OCL constraints. The users must limit the search space by explicitly indicating the number of objects in each class, the number of links of each association and the possible values of each attribute. When the tool cannot find a satisfying instance within the specified search space, this does not mean that the property does not hold, because it can still hold for instances outside that search

[4] http://www-st.inf.tu-dresden.de/oclportal, last visited in July 2014.

[5] We note that the use of these tools was eased by the output of our tool [21] that maps OCL to FOL being SMT-LIB [8] standard.

space (and the user may try to verify the property with wider intervals). In the same vein, the tool UML2Alloy [5] performs bounded verification in relational logic. Also, the extension of USE tool with relational logic for satisfiability checking [19]. From this tool we much appreciate as an usability advantage the facility of graphical display of the instances found as object diagrams. Finally, similar analysis can be performed using propositional logic [20], but we could not find the BV-SAT available from a web page, although it is reported as an automatic tool in [17]. Yet, there is a major advantage in our approach thanks to our mapping [14], the use we make of SMT solvers supports the OCL 4-valued logic, which is not supported by none of the tools described before.

Although they do not perform fully automatic analysis, we consider complementary tools interactive theorem provers like, e.g. HOL-OCL [10] or the Key tool [9]. The fact that they are not fully automatic impact their use that requires too high mathematical background, precluding them from standard software development practice. As a final remark, we note that HOL-OCL supports the 4-valued logic of OCL.

3 The Core Security Metamodel

In this section we explain the Core Security Metamodel (CSM) which allows the description of the security related knowledge that needs to be considered in the development of secure cloud applications. Reflecting the complexity of the security field, the CSM is a composition of 6 sub-models that address different security expertise subareas. Thus, CSM instantiation is facilitated by these groups of related elements which are displayed in the metamodel with different colors, as shown in Figure 1.[6] Next we describe these sub-models, also to understand how they fit together.

Requirement sub-model (green): it is used to qualify security and certification requirements by means of security valuators, mechanisms and certified services.

Property sub-model (yellow): it is used to describe abstract security properties involved in a security requirement, specifying its attributes and values.

Domain sub-model (brown): it is used to describe the domain or context of the CSM instance, identifying the assets to be protected.

Solution sub-model (pink): it is used to show how the security requirements will be achieved by means of solutions and security mechanisms.

Assurance sub-model (blue): it is used to specify the assurance profile and the certification-related elements that would fulfill the certification requirements.

Service Level Agreement sub-model (light blue): it is used to specify SLA agreements that may affect the security properties.

The CUMULUS engineering process aims not only at supporting experts to express their expertise into a model, but also to orchestrate an automated sorting and processing of that information to make it accessible and useful for non security experts. The effectiveness of this approach heavily relies on the OCL validation system which supports three goals in the CSM instantiation activity:

[6] CSM has been already proved its use for real applications to integrate security mechanisms in high risk environments [23,22], but using a different security engineering process in the context of the SecFutur Project (http://www.secfutur.eu).

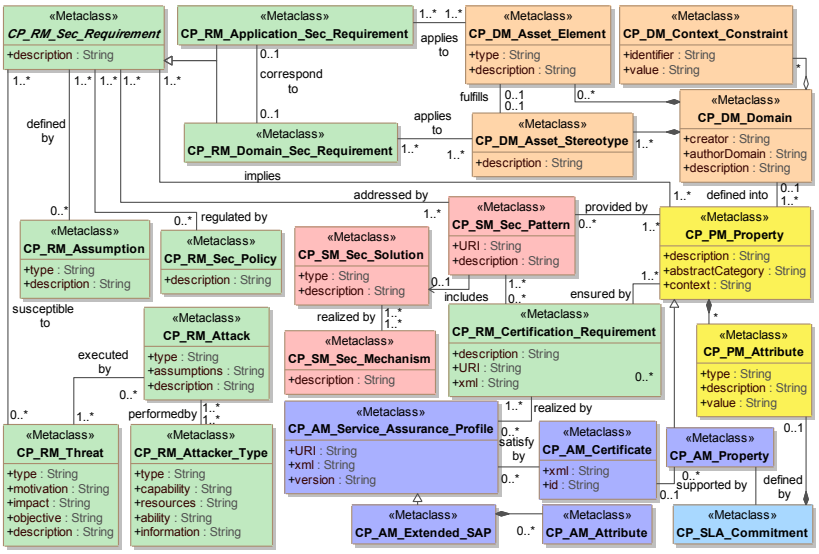

Fig. 1. Core Security Metamodel

1. *Perform an active validation of the modeling process.* This validation raises a warning if the instance does not conform to the metamodel. It also highlights the pieces of information that are missing or wrong. This validation helps experts to avoid wrong specifications that would impact the run time of the system.
2. *Check that required information is present.* It validates whether a valid CSM instance lacks information that is needed by the engineering activities. E.g., transitive association between specific components, empty attributes, etc..
3. *Guide experts during the creation of the CSM instance.* They are guided towards the next piece of information that is needed and its goal in the engineering process.

Therefore the list of OCL constraints is expected to be consistent and reactive enough to support constant interaction with it. Our rules drive an incremental validation system that is gradually triggered within the MagicDraw modelling framework [3].

OCL Constraints. The OCL validation package is composed of 33 rules. Out of these, 27 are structural constraints restraining metamodel associations. Next, we introduce those OCL constraints that do not deal directly with multiplicities.

1. A domain instance must exist and be unique

   inv: `CP_DM_Domain.allInstances()->size() = 1`

2. A certification requirement needs to be associated with a service assurance profile.

   context: `CP_RM_Certification_Requirement` **inv:**
   `(not self.URI.oclIsUndefined()) implies`
   `self.service_assurance_profile->notEmpty()`

3. A certification requirement must be linked directly and through a security pattern to a security requirement and a property

context: `CP_PM_Property` **inv:** `self.certification_requirement->notEmpty()`
`implies self.certification_requirement.sec_pattern.sec_requirement`
`->intersection(self.sec_requirement)->notEmpty()`

4. A certification requirement should be directly linked to a property and a security pattern for that property

   context: `CP_RM_Certification_Requirement` **inv:**
   `self.property->intersection(self.sec_pattern.property)->notEmpty()`

5. An asset stereotype is set up over an asset element that must be considered by an application security requirement of that asset stereotype domain

   context: `CP_DM_Asset_Stereotype` **inv:** `(not`
   `self.asset_element.oclIsUndefined()) implies`
   `self.domain_sec_requirement.application_sec_requirement.`
   `asset_element->includes(self.asset_element)`

6. A security pattern must display a security solution

   context: `CP_SM_Sec_Pattern` **inv:** `(not self.URI.oclIsUndefined())`
   `implies (not self.sec_solution.oclIsUndefined())`

4 Using OCL2FOL to map CSM into First Order Logic

In this section, we first recall our mapping from metamodels and OCL constraints to FOL [13,14,11]. Then, we map the most illustrative constraints introduced in section 3.

- *Type-predicates:* Metamodels' classes are mapped to unary boolean functions. E.g., the class `CP_SM_Sec_Solution` is mapped to $\mathrm{CPSMSecSolution} : \mathrm{Int} \rightarrow \mathrm{Bool}$;
- There are two predicates $\mathrm{isNull} : \mathrm{Int} \rightarrow \mathrm{Bool}$ and $\mathrm{isInvalid} : \mathrm{Int} \rightarrow \mathrm{Bool}$, which return `true` to represent the values `null` or `invalid` (resp.);
- Objects variables are mapped to integer variables, e.g, an object variable cl of type `CP_SM_Sec_Solution` is mapped to an integer variable cl, such that $\mathrm{CPSMSecSolution}(cl)$ holds;
- *Attribute-functions:* Attributes are mapped to integer functions, e.g., the attribute `id` of the class `CP_AM_Certificate` is mapped to a function $\mathrm{CPAMCid} : \mathrm{Int} \rightarrow \mathrm{Int}$.[7]
- *Association-predicates:* Association-ends are mapped, according to their multiplicity, either to predicates or functions. E.g., the association `realizedby` between `CP_AM_Service_Assurance_Profile` and `CP_RM_Certification_Requirement` is mapped into $\mathrm{CPAMSArealizedby} : \mathrm{Int} \times \mathrm{Int} \rightarrow \mathrm{Bool}$.
- For each pair of different classes, e.g. `CP_SM_Sec_Solution` and `CP_DM_Sec_Mechanism` (that are not sub-classes of any other class), the predicates $\mathrm{CPSMSecSolution}$ and $\mathrm{CPDMSecMechanism}$ must be disjoint, i.e: $\forall(x)\ \neg(\mathrm{CPSMSecSolution}(x) \wedge \mathrm{CPDMSecMechanism}(x))$. Similar formulas are included for all type-predicates.
- Also, we map inheritance relations. E.g., in Figure 1, there is an inheritance relation from the parent class `CP_RM_Sec_Requirement` to the children classes `CP_RM_Application _Sec_Requirement` and `CP_RM_Do main_Sec_Requirement`.

[7] For the sake of simplicity, we do not consider attributes with type object, neither multivalued attributes. Also, boolean attributes are always mapped into an integer attribute.

We map this relation as follows: $\forall(x)(\text{CPRMApp}-\text{SecReq}(x) \Rightarrow \text{CPRMSecReq}(x))$ and $\forall(x)(\text{CPRMDomainSecReq}(x) \Rightarrow \text{CPRMSecReq}(x))$.
Since, `CP_RM_SecRequirement` is an abstract superclass, then the following assertion are included: $\forall(x)\ \neg(\text{CPRMAppSecReq}(x) \wedge \text{CPRMDomainSecReq}(x))$ and $\forall(x)(\text{CPRMSecReq}(x) \Rightarrow (\text{CPRMAppSecReq}(x) \vee \text{CPRMDomainSecReq}(x)))$.

- `OCL Boolean`-expressions are translated to formulas, which essentially mirror the logical structure of the OCL expressions, e.g., for the operations `or`, `and`, `implies`, `not`, `notEmpty`, `includes`, `oclIsUndefined`, `forAll`, `exists`, $=$, $\neq$; e.g., `CP_PM_Pro- perty.allInstances()->notEmpty()` is mapped into: $\exists(x)(\text{CPPMProperty}(x))$.
- `OCL Integer`-expressions are basically copied, e.g. $+$, $-$, $*$. Currently, we only cover simple operations (i.e., $=$ and $<>$) over `OCL String`-expressions.
- `OCL Collection`-expressions are translated to *fresh* predicates that augment the signature of the specification. Their meaning is defined by additional formulas also generated by the mapping. E.g., `select`, `collect`, `intersection`, etc..

Next we show the mapping of the CSM constraints numbered 2 and 4 in section 3. [8] Their mapping well represents the one required for the other constraints. Note that, for the constraint 4 two new *fresh* predicates are created: Collect1 and Intersection1.

[2]. `CP_RM_Certification_Requirement.allInstances()->forAll(c|not(c.URI.oclIs-Undefined()) implies (not s.service_assurance_profile->notEmpty()))`

$$\forall(x)(\text{CPRMCertificationRequirement}(x) \wedge \neg(\text{isNull}(\text{CPRMCRurl}(x)) \vee \text{isInvalid}(x)) \Rightarrow \exists(y)(\text{CPAMServiceAssuranceProfile}(y) \wedge \text{CPRMCRrealizedby}(y, x)))$$

[4]. `CP_RM_Certification_Requirement.allInstances()->forAll(c|c.property->in-tersection(c.sec_pattern->collect(p|p.property))->notEmpty())`

$$\forall(x)(\text{CPRMCertificationRequirement}(x) \Rightarrow \exists(y)(\text{Intersection1}(\text{x}, \text{y})))$$
$$\forall(x, y)(\text{Intersection1}(x, y) \Leftrightarrow (\text{CPPMPensuredBy}(y, x) \wedge \text{Collect1}(x, y)))$$
$$\forall(x, y)(\text{Collect1}(x, y) \Leftrightarrow \exists(z)(\text{CPSMSecPattern}(z) \wedge \text{CPSMSPcertification}(z, x) \wedge \text{CPPMPprovidedBy}(y, z)))$$

5 Core Security metamodel validation and instance generation

In this section we explain the analysis that we perform on the OCL constrained CSM metamodel once it is translated to FOL.[9] We first tried to check whether the OCL constraints imposed on the CSM were or not unsatisfiable (and generate an example in the latter case) by feeding them to the SMT solvers Z3 [15] and CVC4 [7]. However, after more than 3 hours running, they did not return any result, and we decided to stop them. We know that this lack of result from Z3 and CVC4 is due to the fact that current techniques for dealing with quantified formulas in SMT are generally incomplete. In particular, they usually have problems to prove the unsatisfiability of a formula with

[8] We want to note that our mapping is not yet complete but it does cover a sufficiently significant subset of the OCL language.

[9] Translation available at http://www.software.imdea.org/~dania/tools/csm.html.

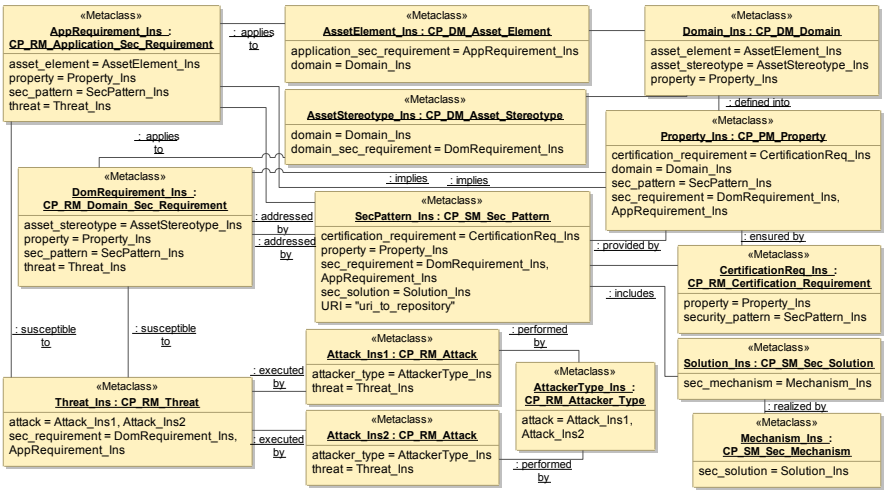

Fig. 2. Automatically generated instance of the security metamodel presented in the Figure 1.

universal quantifiers (our specification is plenty of them).[10] Then, we decided to employ CVC4 as a finite model finder on our specification to check its satisfiability because the input required by it is the same input for the SMT solvers. CVC4 performed a bounded checking and succeeded by returning *sat* and automatically producing finite instances that conform to the OCL constrained CSM. Let us note that to work with the finite model finder CVC4, since the output of our tool [21] is SMT-LIB, we only needed to change in our mapping the sorts Int by a finite sort U. CVC4 run less than 30 seconds to answer SAT and return a simple CSM instance.

Then, we included additional OCL constraints to require a defined URI for all instances of `CP_SM_Sec_Pattern`, to contain a minimum of two `CP_RM_Attack` instances, and at least one instance of each of the following classes: `CP_RM_Attack_Type`, `CP_RM_Certification_Requirement`, `CP_SM_Sec_Solution` and `CP_SM_Sec_Mechanism`. They ensure that generated instances contain at least a minimum amount of information that makes them meaningful for a security expert. Then, we run CVC4 again with these additional constraints, and after less than 1 minute, the instance that we depict in Figure 2 was returned. The instances so obtained with CVC4 match structurally those obtained following the security engineering process and would allow to skip some of its steps (provided that we could automatically tailor the instances obtained by CVC4 to serve as inputs for the modeling framework). As we show next, these instances can be enhanced with knowledge (semantics) from the security domain so as they can serve as input for subsequent steps of the security engineering process.

6 Security enhanced CSM instances

As we already mentioned, the CSM is part of an assisted methodology, supported by the CUMULUS modelling tool [2], that has been initially conceived to take advantage of

[10] More specifically, in this case the problem is introduced by how we map certain types of association ends into FOL. If we do not include their translation, the SMT solvers terminate, but the instances they return are not always valid instances.

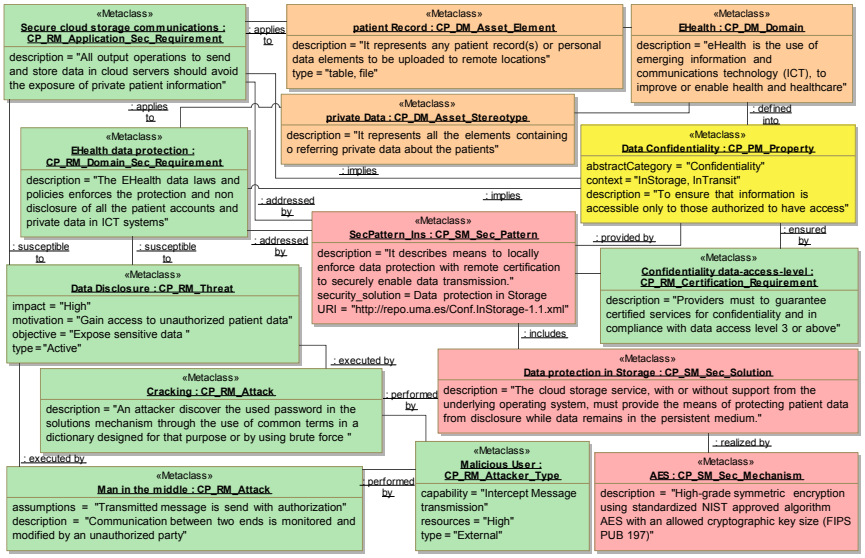

Fig. 3. Domain Security Metamodel

the multiple capabilities provided by the MagicDraw framework [3], particularly of its OCL validation engine. This methodology aims at supporting security experts to specify and communicate to system engineers how to solve security issues for cloud applications. When security experts design their models, i.e., CSM instances, the CUMULUS framework guides the construction of these instances (Domain Security Metamodels-DSMs) with the OCL rules that are continuously validated over them, raising warnings that claim for mandatory elements that are not yet present or errors. This process establishes a common format for the knowledge modeled, ensuring its applicability later on. The resulting instance (i.e., a DSM) is a validated artifact ready to transform security requirements into certification requirements and links to the solutions and mechanisms able to assure local system architectures and their interaction with cloud platforms [6].

For example, the rule [1] in section 3 requires a unique domain instance. Experts dealing with security knowledge in the *EHealth* domain in cloud environments may describe a model for non security experts so as to improve a health care process (we follow Fig. 3). Domain specification is critical to upload DSMs into the appropriate repository, to classify the DSM content adequately. Once a valid domain instance has been created, the validation system triggers those rules that are not yet satisfied so as the model has to be extended to fulfill them. In our example, the framework requests at least one `Property` and one `Asset_Stereotype` instance to be linked with the `Domain`, stemming from the CSM multiplicities and the constraint descriptions. Our DSM is extended with *private data* as an asset stereotype to *represent all the elements containing private patient data* and the security property *Data Confidentiality* (that would *ensure that information is accessible only to authorized users*). This modus operandi is repeated until DSM fully conforms structurally to the CSM and its OCL constraints.

For the sake of space, we do not describe here in full the DSM creation process. But we further describe the DSM instance in Fig. 3. It contains as security requirements

EHealth data protection and *Secure cloud storage communications*, both associated to the threat *Data Disclosure*. In addition, we have created an additional asset *patient record*, potential attacks as *Cracking* or *Man in the middle* and, finally, a common attacker type *Malicious User*. Probably, the most important part of a DSM is the selection of security patterns and certification requirements. The issue to be solved is described in the pattern, in our example, *means to locally enforce data protection with remote certification to securely enable data transmission*. How it should be guaranteed is specified by the certification requirement, in our example, *the usage of certified services for confidentiality and in compliance with data access level 3 or above*. Both plain descriptions have consequences in the security engineering process because they limit the solutions to be deployed for cloud applications. Recalling subsection 3, the last constraint requires that for a security pattern and a solution to be linked, the URI attribute of the pattern must be defined. This constraint demands intervention of the security expert since they search and select from existing repositories, through an API provided by the framework, a suitable pattern that also links a target solution, e.g., *Data protection in Storage* and a security mechanism, e.g., *AES*. As a result of the modelling process, security experts provide a complete artifact ready to fulfill security requirements addressing both the local mechanisms and the remote certification requirements.

Finally, we remark that both instances shown in Figures 2 and 3 resp., are structurally identical. Thus, the engineering process receive a shortcut from the use of automatic finite model finders that ease the path and reduce the time required to build instances since they can automatically generate them. Then, instances can be enhanced with security domain specific knowledge and trigger subsequent engineering activities.

7 Conclusions and Future Work

In this paper we have introduced a security metamodel (CSM) that is constrained by 33 OCL rules that drive the engineering of secure cloud applications. We formally analyzed this metamodel, that is both complex and large, and its constraints, to gain confidence on their consistency and adequacy for the engineering process. We used our previous work, OCL2FOL [13,14,21] to automatically map the metamodel and its constraints to first order logic. Then, we employed successfully a finite model finder, CVC4, that returns '*sat*' for the resulting specification. We also illustrated how the instances automatically generated by CVC4 conform to the CSM and its constraints, and are enhanced with domain security knowledge to get ready to trigger the remaining engineering activities. The automated approach generates an instance that matches one obtained following the engineering process. Based on these results we can say that our formal analysis besides providing higher assurance of the adequacy of the CSM and its rules, also reduces the time and effort required from the security experts in the initial stage of the CUMULUS engineering process. Particularly, since the automatic valid instances generation. Yet, we will need to implement a converter from the CVC4 instances to a valid model input format for MagicDraw to automate the process based on instance generation.

Acknowledgement. This research was partially supported by the 7th EU Framework Programme project CUMULUS (Certification infrastructure for multi-layer cloud services) grant no. 318580 and PARIS (Privacy Preserving Infrastructure for Surveillance)

grant no. 312504, and by the Spanish Ministry of Economy and Competitiveness Project "StrongSoft" (TIN2012-39391-C04-04).

References

1. CUMULUS Project. http://cumulus-project.eu/.
2. D4.2: Tools supporting CUMULUS-aware engineering process v1. http://cumulus-project.eu/index.php/public-deliverables.
3. MagicDraw Modelling Tool. http://www.nomagic.com/products/magicdraw.html.
4. PARIS Project. http://www.paris-project.org/.
5. K. Anastasakis, B. Bordbar, G. Georg, and I.Ray. UML2Alloy: A Challenging Model Transformation. In *MoDELS 2007*, volume 4735 of *LNCS*. Springer, 2007. Tool available at http://www.cs.bham.ac.uk/~bxb/UML2Alloy/download.php, last access: June 2014.
6. M. Arjona, R. Harjani, A. Muñoz, and A. Maña. An Engineering Process to Address Security Challenges in Cloud Computing, 3rd ASE International Conference on Cyber Security, 2014.
7. C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli. CVC4. pages 171–177, 2011.
8. C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0. In *Proc. of the 8th International Workshop on Satisfiability Modulo Theories*, 2010.
9. B. Beckert, U. Keller, and P. H. Schmitt. Translating OCL into First-order Predicate Logic. In *In Proc. of VERIFY Workshop at Federated Logic Conferences (FLoC)*, 2002.
10. A. D. Brucker and B. Wolff. HOL-OCL: A Formal Proof Environment for UML/OCL. In *FASE 2008*, volume 4961 of *LNCS*. Springer, 2008.
11. F. Büttner, M. Egea, and J. Cabot. On Verifying ATL Transformations Using 'off-the-shelf' SMT Solvers. In *MoDELS*, volume 7590 of *LNCS*, pages 432–448. Springer, 2012.
12. J. Cabot, R. Clarisó, and D. Riera. UMLtoCSP: a tool for the formal verification of UML/OCL models using constraint programming. In *ASE 2007, Proc.* ACM, 2007. Tool available at http:/gres.uoc.edu/UMLtoCSP/.
13. M. Clavel, M. Egea, and M. A. García de Dios. Checking Unsatisfiability for OCL Constraints. *Electronic Communications of the EASST*, 24:1–13, 2009.
14. C. Dania and M. Clavel. OCL2FOL+: Coping with Undefinedness. In *Proc. of the MODELS 2013 OCL Workshop*, volume 1092 of *CEUR Workshop Proceedings*, pages 53–62, 2013.
15. L. Mendonça de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and J. Rehof, editors, *TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.
16. C. A. González, F. Büttner, and Jordi Cabot. EMFtoCSP: A Tool for the Lightweight Verification of EMF Models. In *FormSERA*, pages 44–50, 2012. Tool at https://code.google.com/a/eclipselabs.org/p/emftocsp/.
17. C. A. González and J. Cabot. Formal verification of static software models in MDE: A systematic review. *Information & Software Technology*, 56(8):821–838, 2014.
18. R. Harjani, M. Arjona, A. Muñoz, and A. Maña. Towards an Engineering Process for Certified Multilayer Cloud Services, Layered Assurance Workshop. ASAC. 2013.
19. M. Kuhlmann, L. Hamann, and M. Gogolla. Extensive Validation of OCL Models by Integrating SAT Solving into USE. In *TOOLS 2011*, volume 6705 of *LNCS*, pages 290–306. Springer, 2011. Tool available at http://sourceforge.net/projects/useocl/.
20. M.Soeken, R.Wille, and R.Drechsler. Encoding OCL Data Types for SAT-Based Verification of UML/OCL Models. In *TAP*, volume 6706 of *LNCS*, pages 152–170. Springer, 2011.
21. OCL2FOL Project, 2012. http://www.actiongui.org, see OCL2FOL and OCL2FOL+.
22. J. F. Ruiz, A. Maña, M. Arjona, and J. Paatero. Emergency Systems Modelling using a Security Engineering Process. In *Proc. of 3rd Int. Conf. SIMULTECH*. SciTePress, 2013.
23. J.F. Ruiz, A. Rein, M. Arjona, A. Maña, A. Monsifrot, and M. Morvan. Security Engineering and Modelling of Set-Top Boxes. In *Proc. of ASE/IEEE BioMedCom*, 2012.