Supporting Security Engineering at Design Time with Adequate Tooling

Jörn Eichler, Andreas Fuchs, Nico Lincke
Fraunhofer Institute for Secure Information Technology (SIT)
Darmstadt, Germany
Email: {joern.eichler,andreas.fuchs,nico.lincke}@sit.fraunhofer.de

***Abstract*—Security engineering is considered to be a challenging task in order to build systems that remain dependable in the face of malice, error, or mischance. Recent approaches propose the application of domain specific modeling languages (DSMLs) in order to facilitate security engineering activities. To support the development and application of adequate DSMLs, agile approaches and frameworks to provide appropriate tooling are needed. In this paper, we document our experiences developing modeling tools for two different DSMLs in the domain of security engineering. We sketch the language and implementation requirements for our modeling tools, design and implementation considerations, and report on pitfalls and remaining issues with regard to the development of modeling tools based on our experiences.**

I. Introduction

The security analysis of information technology (IT) systems and the design of secure IT systems is and has always been regarded as a complex and laborious challenge [1]–[3]. In order to support architects, developers, and security engineers tackling this challenge, the application of formal methods that already solved major challenges in other fields of computer science has become a common strategy, e.g., [4], [5]. Similarly, informal approaches have been developed to assist in building secure systems and are now widely used (cf. [6], [7]). Recent approaches propose the application of domain specific modeling languages (DSMLs) to further facilitate security analysis and design and to integrate security issues in current model-driven engineering approaches, e.g., [8], [9].

Although several approaches with supportive tooling have been proposed, literature and our own experience suggest that only few approaches for systematic security engineering supported with appropriate tooling are applied in industry [10]. One reason for this lack of application might be an insufficient provision of comprehensive and adequate modeling support by current approaches for security engineering [11].

This situation might be due to the fact that the development of DSMLs and respective modeling tools in the domain of security engineering is itself a challenging task. Stakeholders from different domains need to be involved in the development that do not always share the same background and perspective on security. In the context of model-driven engineering, model-driven development of tooling for the application of DSMLs is becoming common practice [12]. We understand this situation – the unsatisfactory application of tool-based security engineering and the commodification of model-driven development practices with the purpose of providing DSML tooling – as an opportunity. The application of agile development approaches and frameworks for model-driven development of DSMLs and respective tooling might support the provision of adequate tooling for security engineering at design time.

To our best knowledge, experiences in the development of modeling tools to facilitate security engineering have not been documented so far. Therefore, we contribute with our paper a report on our approach to develop DSMLs and tooling for two different proposals for security engineering at design time and detail some pitfalls and remaining issues from our experiences.

The remaining paper is structured as follows. The next section covers some background information on security engineering and presents related work on representing security statements and respective tooling. Section III introduces briefly our approaches for security engineering at design time and details important goals and requirements for the development of adequate tooling for those approaches. Our development approach and implementation of our tooling is elaborated in Section IV. Section V discusses our experiences, pitfalls, and remaining issues with regard to our development approach and frameworks applied. A closing conclusion summarizes the results and indicates some future work.

II. Background and Related Work

This section presents some background information on security engineering and reports on related work that introduces concepts for representing security statements or developed tools to facilitate this task.

A. Security Engineering

Multiple definitions of security have been proposed, e.g. [13], [14]. We understand security with regard to IT systems as a property of a system to take only those states that do not violate the security goals of the system. Security engineering is considered to be *"about building systems to remain dependable in the face of malice, error, or mischance"* [3]. Thus, security engineering focuses all security-related activities in the life cycle of a system and provides methods to construct and maintain the security of the system.

From literature, security engineering encompasses methods and techniques that might be subsumed as security principles, patterns, formal modeling and verification methods, and process models. Security principles are generalized experiences

to engineer secure systems in order to conquer this difficult task. A prominent example are the design principles for secure systems from Saltzer et al. [1]. Security patterns have been introduced to security engineering focusing solutions. They describe a recurring security problem in a given context and present a generic solution for it. Common patterns are provided by [7], [15]. Formalization of security problems is an established practice in security engineering. Formal approaches to security generally build upon a system model, a threat model, and a formal definition of the security properties in question [16]. Common problem areas tackled with formal methods in security engineering include access control, information flow, and cryptographic protocols (cf. [17]–[19]). Process models for security engineering integrate security-related activities in the life cycle of a system and provide guidance on how to apply aforementioned methods and techniques systematically. Examples for such process models are provided by [20], [21].

B. Modeling Tools

It is a common approach to support security engineering methods and techniques with supportive modeling tools. General strategies to provide modeling tools in the domain of security engineering are the application of existing (general purpose) tooling, adaptations of those general tools, and the provision of dedicated tooling. User interfaces for such modeling tools are realized using graphical and textual representations and editors as well as form-based approaches. Generally, the development approach taken to implement those tools is neither documented nor obvious.

With a focus on risk analysis and assessment, AURUM follows a form-based approach that supports users from the collection of resources, threats, and controls to the evaluation of residual risks [22]. Mouratidis et al. introduce with Secure Tropos an extension of the general Tropos methodology that is supported by a dedicated tool with graphical representation [23]. Analogous, Elahi et al. present an extension of the *i** approach accompanied with a graphical editor for the respective models [24]. A graphical DSML and dedicated editor for risk analysis in a model driven tool chain is provided by Normand et al. in [25]. A textual approach is taken by Dimkov et al. representing and analyzing attack scenarios [26]. Other textual approaches include the SH-Verification Tool [27] that uses textual descriptions for system modeling and checking system conditions. Further on, it utilizes dialects of Temporal Logics [5] that are also used as textual property languages in many other security tools. A combination of a well-established method, a general graphical tooling with slight methodological extensions, and a form based interface is provided by Microsoft[1]. Extending Data Flow Diagrams (DFDs) with security annotations that can be drawn with a stripped and tailored Visio integrated in a form-based interface, the tool supports systematic threat identification, assessment, and development integration [6].

[1]http://www.microsoft.com/en-us/download/details.aspx?id=2955

Two approaches might be situated between the application of general tooling and the provision of dedicated tooling. With CORAS – an approach for model-based security analysis – a Profile for the Unified Modeling Language (UML) has been introduced [28]. Later, dedicated graphical tooling has been provided to support CORAS more efficiently [29]. Similarly, SecureUML reused initially general purpose UML tooling to specify access control policies and extended later on an existing tool to provide with SecureMOVA enhanced analysis capabilities [8], [30].

The application of general purpose UML tooling has been and is still common in the domain of security engineering. A well known approach is UMLsec that provides an UML Profile to specify security requirements and assumptions [31]. Likewise, a model-driven security engineering framework for workflows called SECTET uses with SECTET-UML an UML Profile to specify trust properties [32]. Further examples for approaches that use UML Profiles to support security engineering are [9], [33]. Other general concepts for describing system behavior graphically in the context of security analysis include Petri nets [4] and its many derivatives as well as Strand Spaces [34].

Unfortunately, none of these modeling tools provides support for the DSMLs in question (SeMF or SecEML, cf. subsections III-B and III-C). Therefore, we decided to implement corresponding tooling as it is presented in the following sections.

III. Language and Implementation Requirements

This paper presents the approach and experience gained during the development of tooling for two different DSMLs that are both targeted at security. This section provides goals and requirements for the development of adequate tooling to support the application of those DSMLs. Subsection III-A sketches shared goals for both DSMLs that highlight important issues with regard to the development of our tooling. Subsections III-B and III-C provide short overviews to the DSMLs and their specific requirements.

A. General Requirements

As we have detailed in the preceding section II-B, several approaches have been presented in the past to support security engineering by applying model-based methods supported with respective tooling. We also mentioned a lack of application of most of those methods in the field.

To allow for a better alignment of requirements for the application of modeling methods in the field and tailored DSMLs as well as their tooling we strive for an approach that allows for short feedback cycles between the development of a DSML, respective tooling, and its application in controlled environments as well as in the field. Therefore, we identify the following requirements:

- Simple, yet rigorous, specification, application, and modification of the DSML's abstract and concrete syntax
- Easy mapping of abstract and concrete syntax

- Supporting (model-driven) framework to avoid boilerplate coding and incremental development (cf. [12])

Besides these requirements to allow for short feedback cycles, further considerations need to be taken into account when developing a DSML to support security engineering:

- Extensibility for new expressions and constraints, preferably supported by existing languages and tooling, e.g. Object Constraint Language (OCL)
- Interoperability with existing software engineering modeling tools
- Compatibility with well established GUI concepts

B. SeMF

The Security Modelling Framework SeMF [35] provides a formal language based semantics for expressing security statements in a comprehensible way. Through these means, it does not only provide an axiomatic framework, but rather an underlying reasoning framework that allows for unambiguous security statements. Further, SeMF distinguishes itself from other formal frameworks by providing some characteristics that make it especially suitable for application in the context of security engineering [36]. We now give a quick introduction to some of the concepts of SeMF to demonstrate the advantages gained by specialized DSMLs and authoring tools.

The framework SeMF provides a two-fold contribution. First, it is a semantic framework that utilizes formal language theory and alphabetic language homomorphisms to express security statements as *positive statements*. These refer to the *security guarantees* given by a system which resembles the set of *all possible behavior* minus the *security failures*. These semantics are based on the concepts of agents $\mathbb{P}$, actions Σ, traces of actions Σ^*, local view of agents λ_P and agents' initial knowledges W_P.

Second, SeMF provides a set of predefined properties as predicates that base on these semantics. In order to describe a given system's security aspects, these predicates formulate a common vocabulary. A subset of properties to be used within this paper's example can informally described as follows:
$\mathbf{precede}(\mathbf{a}, \mathbf{b})$ holds if whenever action b occurs, action a must have occurred (before) as well.
$\mathbf{auth}(\mathbf{a}, \mathbf{b}, \mathbf{P})$ holds if whenever action b occurs, action a must have happened authentically for agent P, according to P's *local view* and *initial knowledge*.
$\mathbf{conf}(\mathbf{A}, \mathbf{p}, \mathbf{M}, \mathbf{W})$ holds ..., $\mathbf{trust}(\mathbf{P}, \mathbf{prop})$ holds ..., ...

In order to express the security aspects of a given system formally, a SeMF-based representation consists of the set of agents and their actions within the system: $\mathbb{P} = \{PC_1, PC_2, \ldots\}, \Sigma = \{send(PC_1, msg), \ldots\}$. Properties and Assumptions are defined based on these sets utilizing SeMF's semantics, e.g. $precede(\{send(PC_i, msg) \mid i \in \{1, 2\}\}, recv(PC_3, msg))$ (any message that PC_3 receives has been sent by PC_1 or PC_2). The tools used in this regard are typically pen & paper or LaTeX.

For the utilization in practice (especially within complex and advanced systems) there are however three challenges to be solved or improved, in order to further facilitate adoption:

- Assistance w.r.t. valid expressions (typing, iterators, ...)
- Increase of efficiency (auto-completion, refactoring, ...)
- Support for broader audience (usability, guidance, ...)

C. SecEML

The Security Engineering Modeling Language (SecEML) is based on a language for lightweight modeling of security concepts [37]. It is closely integrated in a security engineering process model for electronic business processes called SecEPM [38].

SecEML is designed to capture artifacts from SecEPM activities in order to support subsequent activities in the security engineering process, to check the validity of the security problem formulation and the security design decisions, and to derive general software engineering artifacts like documentation, configuration artifacts, and test cases. SecEML targets not so much security professionals but business process experts and developers. Therefore, SecEML provides a set of general security engineering concepts and their relations as well as constraints and expressions on those elements to support the security engineering activities. Figure 3 shows the SecEML workbench and an example for a SecEML model.

With this background we identified the following additional requirements for the development of respective tooling:

- Integration in Business Process Management (BPM) tool chain (e.g., alignment with languages applied, versioning, workbench integration)
- Modular partitioning of models (e.g., separation of threat class catalogs from actual threat instances)
- Technical assistance in the creation and validation of models (e.g., understandable and focused error marking)

IV. Development Approach and Implementation

Based on our requirements and the main objective to align application requirements and conditions of modeling tools with a tailored DSML and respective tooling we chose a feature-driven agile (feature-oriented iterative and incremental) development approach utilizing model-driven techniques for our tooling.

First, we analyzed existing frameworks for model-driven development and existing, extensible editors that might serve as a starting point. We identified three candidate frameworks out of our survey, namely ArgoUML[2], Eclipse Graphical Modeling Framework (GMF)[3]/Graphical Editing Framework (GEF)[4], and Eclipse Xtext[5].

For the SeMF approach, the first implementation has been based on ArgoUML and utilized UML Profiles for usability reasons (cf. [39]) in order to provide a graphical representation [40]. However, this implementation displayed a couple of drawbacks:

- The UML syntax did not support what was needed; namely the possibility to reference attributes or member

[2]http://argouml.tigris.org/
[3]http://www.eclipse.org/modeling/gmp/
[4]http://www.eclipse.org/gef/
[5]http://www.eclipse.org/Xtext/

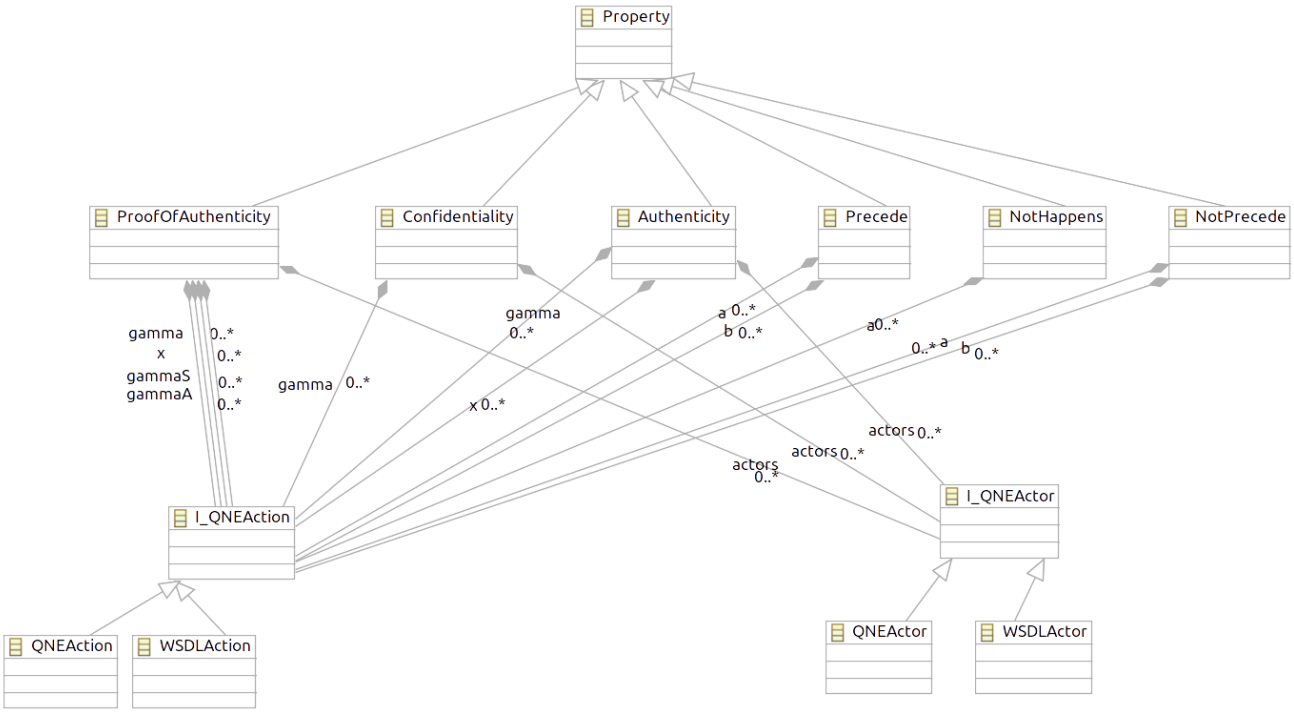

Fig. 1. xSeMF Ecore metamodel extract

methods within a given class. An extension would have been possible but voided the advantage of a least effort implementation.

- Especially for non-trivial or larger models the graphical representation became too complicated too quickly, such that even decomposition techniques did not yield positive results.

An early implementation of SecEML has been based on GMF/GEF to provide a graphical representation of the security analysis and design models. GMF/GEF was favored over ArgoUML in order to offer more degrees of freedom to customize the concrete syntax. Unfortunately, the approach exposed several nuisances:

- As in the case with the first implementation of the SeMF approach, the graphical representation got quite complex quickly, domain users (i.e., business process experts) confronted with the graphical representation expressed unease and confusion.
- The model-driven tool chain demonstrated rising complexity after some iterations changing the abstract syntax, especially in order to maintain the mapping to the concrete syntax and the functioning of the self-developed extensions to the generated editor components.
- Adaptations to the concrete syntax yielded rising efforts and reduced the benefits of the model-driven tool chain.

Based on this experiences, it was decided for SeMF to implement a textual representation called xSeMF using the Xtext language framework. The basis for the language development was the predicate-based SeMF expressions used within formal proofs. A first version of a grammar was implemented to represent these notations and gain familiarity with the Xtext framework. From the grammar the Ecore metamodel and a basic editor has been generated and adapted – especially the scoping providers. Even in this first version, some custom scoping needed to be added in xSeMF as there were three semantically different ways to scope within the "Actor" hierarchy (Actors have SubActors, SuperActors and Actions; both of the former are themselves Actors again and the latter requires for type-inferred parameters). These custom scopings remain until the current version, where beyond the qualification of names a (method-like) signature mapping of parameters needs to be performed Actions.

For SecEML we switched back to Xtext as we had gained some experience before with a similar language [37]. We utilized the model-driven tool chain of the framework by providing the grammar first. The concrete syntax resembles the Human-Usable Textual Notation (HUTN) for familiarity but with less syntax elements following design rules discussed in [41]. We generated the Ecore metamodel and the basic editor from the grammar and implemented incrementally additional productivity features.

Based on these first versions, we evolved the DSMLs as well as the editors in several iterative cycles. These iterations were based on lessons learned, Ecore metamodel analysis (cf. section V) and expert discussions.

As an example, in xSeMF a type-system for abstract representations of type inheritance and instantiation without actual definitions and instantiations was developed – $Type\ NaturalNumber\ is\ Number$; $Number\ n1$ – and improved several times.

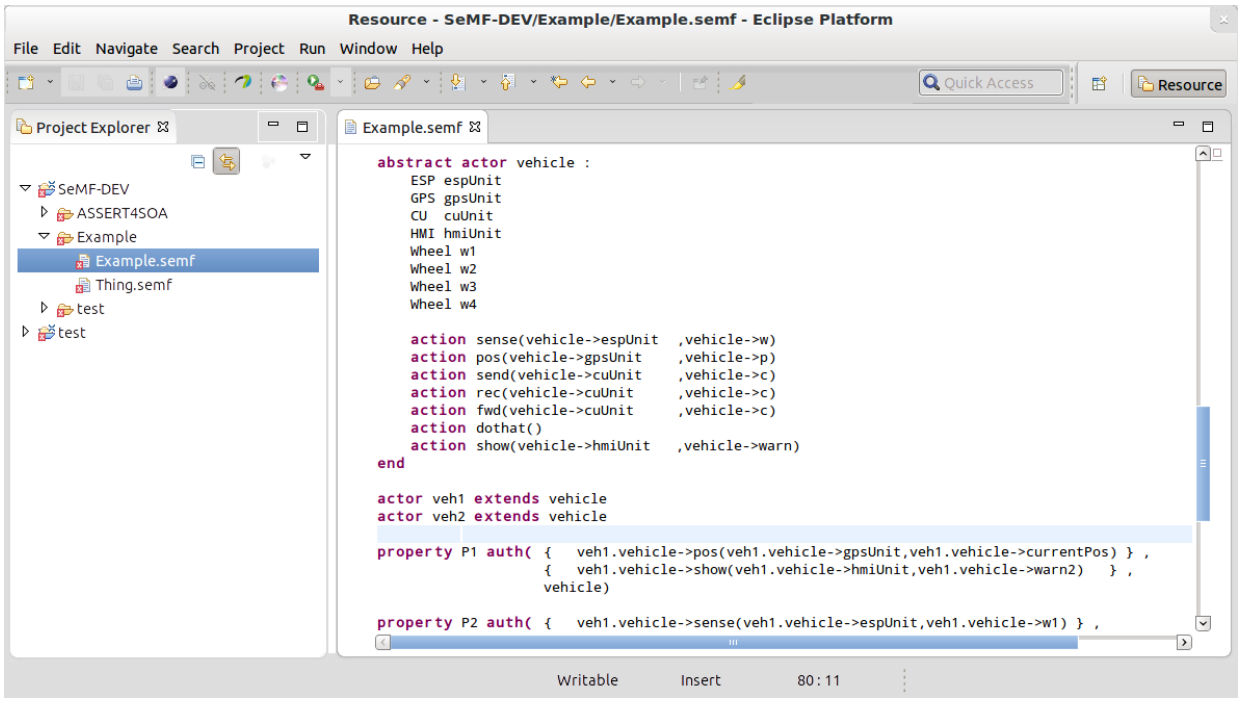

Fig. 2. xSeMF workbench

For SecEML two challenges influenced our development approach in particular: the application of general expressions on SecEML models for validation and analysis as well as the implementation of cross-language references in SecEML models. In order to maximize the applicability of general expressions on SecEML models, OCL has been selected as expression language. This way expressions could be integrated directly in the Ecore metamodels as constraints or properties as well as they could be used for, e.g., the identification of quick-fix candidates in the editor or ad-hoc queries on SecEML models using the Eclipse OCL Console[6]. To allow for a holistic modeling perspective impact propagation, cross-language references, and further useful features have been implemented for SecEML. Both decisions forced to change the application of the tool chain: the Ecore metamodel could not longer be generated, but a polished and enhanced metamodel is now maintained separate from the grammar for the concrete syntax.

As a result, the (still in development) xSeMF language provides an editor with the auto-completion features of Xtext's scoping. This language provides:

- An abstract type system with multiple inheritance
- An object system for "Actors" with multiple inheritance and "SubActors" as well as parameterized "Actions" as members
- A construct for qualified name referencing for actions, actors and parameters as well as type inference on the latter (currently in development)
- The notion of properties utilizing these system representation syntaxes

A screen shot of the editor is presented in Figure 2 and an extract of the derived Ecore model can be found in Figure 1.

Main features of the resulting SecEML editor are:

- Inter-language type-safe cross references and annotations of respective artifacts like business process models
- Evaluation of expressions on cross-referenced models (validation and analysis)
- Semantically checked auto-completion and quick-fixes
- Tight integration in the BPM tool chain through auto-updates on resulting transformations

V. Discussion

The presented implementations of security concepts using textual DSMLs appear suited in the current state. The first implementation approaches using graphical representations suffered from lack of features, overview, and simplicity.

The implementation in a textual representation required some adaptations for SecEML in comparison to graphical representations as for example references on (external, cross-language) model entities always need a human readable, textual identifier that is not always present in other graphical DSMLs.

For xSeMF the textual format did not differ too much from the blueprint of SeMF. However, the necessity to define a type system proved to be more challenging than expected. Also the usage of scoping, qualified naming and type inference (based on these) required a lot of customization work.

The usage of a well-established modeling framework (namely EMF) in combination with a development framework for textual languages proved to be very helpful during

[6]http://www.eclipse.org/modeling/mdt/?project=ocl

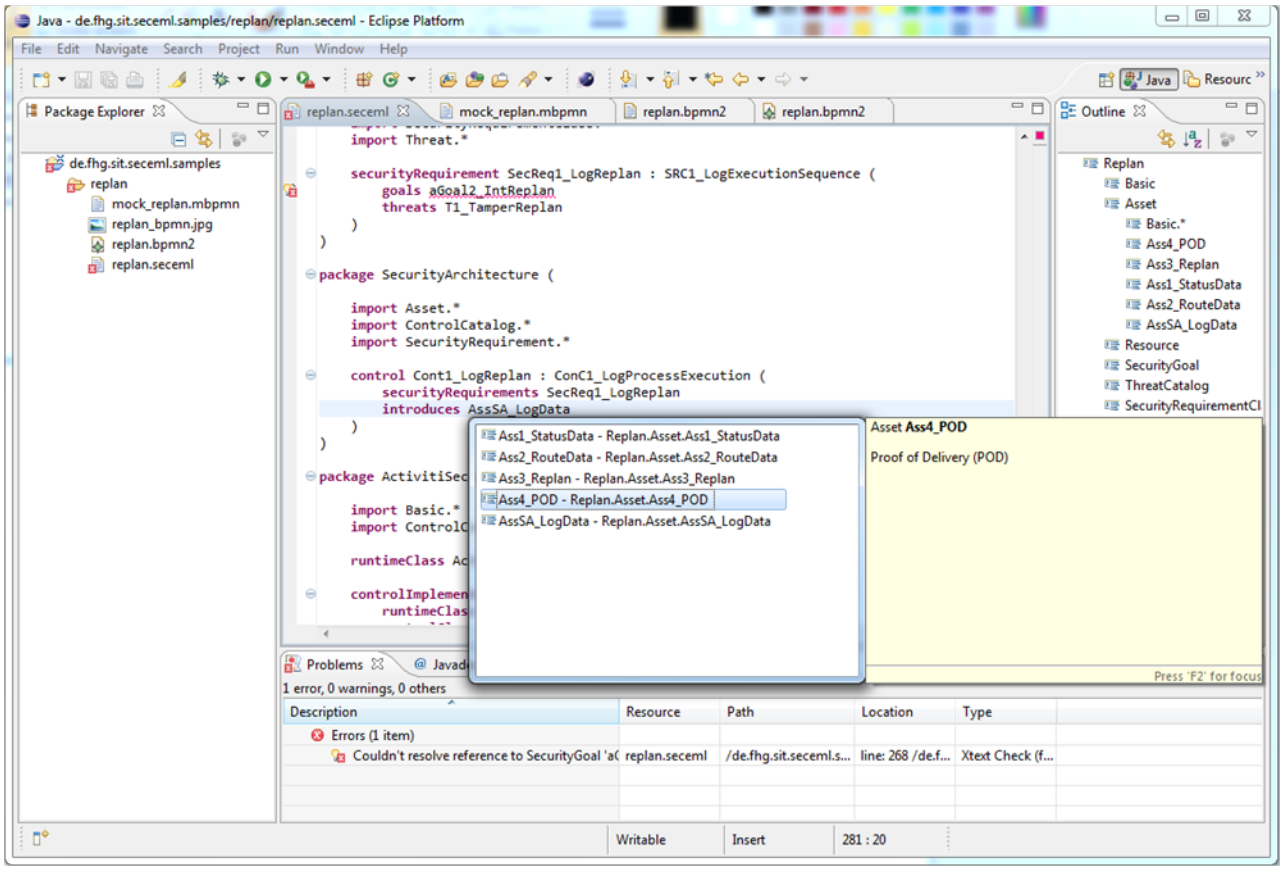

Fig. 3. SecEML workbench

the implementation. In comparison to other approaches (like SHVT [27] using Lisp) a lot of effort for the language generation logics did not have to be redeveloped. However, in comparison to such general purpose languages the lack of a fully equipped inline DSL required some overhead. The two different engineering methods of constructive (grammar) vs. constraining (scoping, OCL) definition of the language space posed additional complexity, that might be mitigated by additional features in the language development framework (Xtext). Especially for SecEML, the interoperability with other DSMLs on the basis of EMF was of great use (and is as well a planned feature for xSeMF). General accessibility of external model entities by general (and generated) means in the first place and iterative refinement (and incremental implementation) with regard to supporting language and tooling features suited very well to our development approach.

Regarding usability, the comparison between SeMF and xSeMF in the way that systems and security properties are described reveals advantages as well as disadvantages especially w.r.t. (syntactic) enforcement of the developed type system. Whilst in the case of pen & paper an analyst can focus on the reasoning about properties, xSeMF forces him/her to first correctly and consistently define the type system and system model. This comes as an advantage and disadvantage equally. Because the flexibility of mathematical expressions (that utilize human context awareness) is unavailable, an analyst needs to spend more time with the construction of the system model. At the same time however, the integrated syntax checker (that is a projection of semantic rules) prevents that some errors may slip by – especially in large models. Similarly, the lack of mathematic expressions ($\forall$, $\exists$, Greek letters, ...) requires more writing and an unnatural usage by experts in formal methods. At the same time, it might be appreciated that the resulting languages seem to be easier to comprehend by non-experts in formal methods with a programming background.

Other features of our EMF-based implementation enable more advanced engineering techniques. The generation of an Ecore metamodel (and its graphical representation) were leveraged in xSeMF as an analysis tool for the grammar development. Out of the generated Ecore metamodel, structural weaknesses as well as an unfortunate syntax-semantics distance could be evaluated and the grammar could be improved accordingly. These techniques also reduced the amount of necessary custom adaptations and thereby reduced overall complexity of the language implementation. Within the SecEML development, the Ecore metamodel generation was mainly used for rapid prototyping purposes. After the first iterations to stabilize the basic grammar, the Ecore metamodel was customized and polished in order to allow for good integration with OCL expressions and external language integration. Then,

the final grammar was built on top of that (handcrafted) Ecore metamodel.

Some drawbacks of the tooling for these usages however included the distance of the Ecore metamodel to the grammar DSML (regarding, e.g., the lack of minimal bounds and maximal bounds in the Ecore metamodel after generation) as well as the distance from the grammar to the actual language (due to the custom scoping providers, e.g., qualified names or XML Schema and regular expression like features for minimal and maximal bounds for entity relations "{min,max}" rather than $+, ?, *$).

Some kind of relations (e.g., implications between properties in xSeMF, major relations between model entities) may be better represented in a graphical way. For the future, the authors will investigate the possibility to replace the current implementation with a hybrid approach that combines a graphical representation with a textual representation depending on the level of granularity and detail.

VI. Conclusion

This paper presented our development approach and the implementation of tooling for two DSMLs to support security engineering at design time. In particular, we discussed the experience gained during the development of SecEML and xSeMF (still in progress). The developed language aware editors demonstrate the suitability of the chosen approach. The comprehensibility and overview increased drastically with a textual representation compared to the graphical representations based on UML and a custom syntax. Furthermore, the possibility to maintain short feedback cycles between DSML adjustment, concrete syntax improvement, tool provision, and feedback from stakeholders proved very helpful.

The experiences made during the development of the two language editors are similar and indicate that they may be applicable for other attempts to describe non-executable languages based on this approach as well as the utilized framework. Especially the two different utilizations of inspecting and altering the generated Ecore model – for sanity check and rapid prototyping – may be useful for upcoming projects. In order to facilitate further adoption of our approach for similar use cases, support for enhancements in the DSML for the language development (i.e., the Xtext grammar specification language) to utilize more features of the underlying EMF might be interesting, e.g., the specification of upper and lower bounds or the utilization of OCL constraints.

For the near future, it is planned to finalize the xSeMF language development (to be presented w.r.t. details about the projection of semantic facts to syntactic rules) and development of adapters for inclusion of other system model representations or system model generations, e.g., the Web Services Description Language (WSDL). In the long run, the inclusion within an actual security engineering life cycle is planned.

In the case of SecEML, generalization of the (type-safe and tool supported) integration of other DSMLs is a next step. Furthermore, an evaluation of SecEML applying controlled experiments is planned to supplement our experiences with more empirical evidence. Complete language specification, tooling, and examples are prepared for publication and open access to allow for an independent application and evaluation of SecEML.

Acknowledgment

The work presented was developed in the context of the projects Innovative Services for the Internet of the Future (InDiNet, ID 01IC10S04F) which is funded by the German Federal Ministry of Education and Research as well as TERESA (IST-248410) and ASSERT4SOA (CP-257351) which are funded by the European Union within FP7.

References

[1] J. Saltzer and M. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.
[2] C. Pfleeger, "The fundamentals of information security," *IEEE Software*, vol. 14, no. 1, pp. 15–16, 1997.
[3] R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley & Sons, 2001.
[4] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.
[5] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science*. IEEE, 1977, pp. 46–57.
[6] F. Swiderski and W. Snyder, *Threat modeling*. Microsoft, 2004.
[7] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns: Integrating Security and Systems Engineering*. John Wiley & Sons, 2006.
[8] D. Basin, J. Doser, and T. Lodderstedt, "Model driven security: From UML models to access control infrastructures," *ACM Transactions on Software Engineering and Methodology*, vol. 15, no. 1, pp. 39–91, 2006.
[9] N. Moebius, W. Reif, and K. Stenzel, "Modeling security-critical applications with UML in the SecureMDD approach," *International Journal On Advances in Software*, vol. 1, no. 1, pp. 59–79, 2009.
[10] R. Vaughn, R. Hennig, and K. Fox, "An empirical study of industrial security-engineering practices," *Journal of Systems and Software*, vol. 61, no. 3, pp. 225–232, 2002.
[11] M. Siponen and J. Heikka, "Do secure information system design methods provide adequate modeling support?" *Information and Software Technology*, vol. 50, no. 9-10, pp. 1035–1053, 2008.
[12] J. Gallardo, C. Bravo, and M. Redondo, "A model-driven development method for collaborative modeling tools," *Journal of Network and Computer Applications*, vol. 35, pp. 1086–1105, 2011.
[13] B. Fabian, S. Gürses, M. Heisel, T. Santen, and H. Schmidt, "A comparison of security requirements engineering methods," *Requirements Engineering*, vol. 15, no. 1, pp. 7–40, 2010.
[14] ISO/IEC, "ISO/IEC 13335-1: Information technology - security techniques - management of information and communications technology security - part 1: Concepts and models for information and communications technology security management," 2004.
[15] D. Hatebur, M. Heisel, and H. Schmidt, "A security engineering process based on patterns," in *18th International Conference on Database and Expert Systems Applications (DEXA 2007)*, 2007, pp. 734–738.
[16] J. Bau and J. Mitchell, "Security modeling and analysis," *Security & Privacy, IEEE*, vol. 9, no. 3, pp. 18 –25, 2011.
[17] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.
[18] H. Mantel, H. Sudbrock, and T. Kraußer, "Combining different proof techniques for verifying information flow security," *Logic-Based Program Synthesis and Transformation*, pp. 94–110, 2007.
[19] M. Brusó, K. Chatzikokolakis, and J. Hartog, "Formal verification of privacy for RFID systems," in *Computer Security Foundations Symposium (CSF 2010)*. IEEE, 2010, pp. 75–88.
[20] J. J. Whitmore, "A method for designing secure solutions," *IBM Systems Journal*, vol. 40, no. 3, pp. 747–768, 2001.
[21] D. Mellado, E. Fernndez-Medina, and M. Piattini, "A common criteria based security requirements engineering process for the development of secure information systems," *Computer Standards & Interfaces*, vol. 29, no. 2, pp. 244–253, 2007.

[22] A. Ekelhart, S. Fenz, and T. Neubauer, "AURUM: A framework for supporting information security risk management," in *Proceedings of the 42nd Hawaii International Conference on System Sciences*. IEEE, 2009, pp. 1–10.

[23] H. Mouratidis and P. Giorgini, "Secure tropos: A security-oriented extension of the tropos methodology," *International Journal of Software Engineering and Knowledge Engineering*, vol. 17, no. 2, pp. 285–309, 2007.

[24] G. Elahi, E. Yu, and N. Zannone, "A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities," *Requirements Engineering*, vol. 15, pp. 41–62, 2010.

[25] V. Normand and E. Félix, "Toward model-based security engineering: developing a security analysis DSML," in *Proceedings of the First International Workshop on Security in Model Driven Architecture (SEC-MDA)*, 2009.

[26] T. Dimkov, W. Pieters, and P. Hartel, "Portunes: representing attack scenarios spanning through the physical, digital and social domain," *Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security*, pp. 112–129, 2011.

[27] P. Ochsenschläger, J. Repp, and R. Rieke, *Simple Homomorphism Verification Tool – Tutorial*, Fraunhofer Institute for Secure Telecooperation SIT, Darmstadt, 2002.

[28] F. den Braber, T. Dimitrakos, B. A. Gran, M. S. Lund, K. Stølen, and J. Ø. Aagedal, "The CORAS methodology: model-based risk assessment using UML and UP," in *UML and the Unified Process*, L. Favre, Ed. IRM Press, 2003, pp. 332–357.

[29] F. den Braber, I. Hogganvik, M. Lund, K. Stølen, and F. Vraalsen, "Model-based security analysis in seven steps – a guided tour to the CORAS method," *BT Technology Journal*, vol. 25, no. 1, pp. 101–117, 2007.

[30] D. Basin, M. Clavel, J. Doser, and M. Egea, "Automated analysis of security-design models," *Information and Software Technology*, vol. 51, no. 5, pp. 815–831, 2009.

[31] J. Jürjens, *Secure Systems Development with UML*. Springer-Verlag, 2005.

[32] M. Alam, M. Hafner, and R. Breu, "Model-driven security engineering for trust management in SECTET," *Journal of Software*, vol. 2, no. 1, pp. 47–59, 2007.

[33] A. Rodrguez, E. Fernndez-Medina, J. Trujillo, and M. Piattini, "Secure business process model specification through a UML 2.0 activity diagram profile," *Decision Support Systems*, vol. 51, no. 3, pp. 446–465, 2011.

[34] F. Fábrega, J. Herzog, and J. Guttman, "Strand spaces: Proving security protocols correct," *Journal of Computer Security*, vol. 7, no. 2/3, pp. 191–230, 1999.

[35] S. Gürgens, P. Ochsenschläger, and C. Rudolph, "On a formal framework for security properties," *International Computer Standards & Interface Journal*, vol. 27, pp. 457–466, 2005.

[36] A. Fuchs and R. Rieke, "Identification of authenticity requirements in systems of systems by functional security analysis," in *Workshop on Architecting Dependable Systems (WADS 2009), in Proceedings of the 2009 IEEE/IFIP Conference on Dependable Systems and Networks, Supplementary Volume*, 2009.

[37] J. Eichler, "Lightweight modeling and analysis of security concepts," in *Engineering Secure Software and Systems (ESSoS 2011)*, ser. LNCS. Springer, 2011, vol. 6542, pp. 128–141.

[38] ——, "SecEPM: A security engineering process model for electronic business processes," in *Proceedings of the 9th IEEE International Conference on e-Business Engineering (ICEBE 2012)*. IEEE, 2012, pp. 206–213.

[39] C. Talhi, D. Mouheb, V. Lima, M. Debbabi, L. Wang, and M. Pourzandi, "Usability of security specification approaches for UML design: A survey," *Journal of Object Technology*, vol. 8, no. 6, pp. 103–122, 2009.

[40] J. Bysewski, "An UML-metamodel for formally-based security reasoning," Master's thesis, Universität Siegen, 2011.

[41] L. McIver and D. Conway, "Seven deadly sins of introductory programming language design," in *Software Engineering: Education and Practice (SEEP 1996)*. IEEE, 1996, pp. 309–316.