Versioning and Evolution Requirements for Model-Based System Development

Thomas Ruhroth*, Stefan Gärtner†, Jens Bürger*, Jan Jürjens*, Kurt Schneider†
*Chair of Software Engineering, TU Dortmund, Germany
Email: {thomas.ruhroth,jens.buerger,jan.jurjens}@cs.tu-dortmund.de
†Software Engineering Group, Leibniz Universität Hannover, Germany
Email: {stefan.gaertner,kurt.schneider}@inf.uni-hannover.de

***Abstract*—Long-living software systems "ages" not by wearing out, but by failing to keep up-to-date with its requirements. Moreover, security is an increasingly important quality facet in modern information systems and needs to be retained properly. In model-based system development, this leads to a continuously changing information system model accordingly. The problem is that software engineers cannot simply overview changes of the system model and their impact on the applied security model. To overcome this problem, a semantic representation of model changes is needed which is determined from fine-grained edit operations. Based on the semantic representation of system model changes, software engineers are supported to choose an evolution strategy of the associated security model. In this paper, we discuss challenges and problems that arise from the granularity of the change operations as well as the selection of different evolution strategies which can be performed interleaved.**

I. Introduction

Software is changed to add functionality, to fix problems or to fulfill other requirements. Since every change needs to be managed, there exists a considerable variety of approaches for the evolution of software and models which are specialized in one way or another. Most approaches have in common that they are based on a model of the regarded system. Hence, a challenge is to combine specific approaches to consider different aspects regarding the evolution of the system model. Whenever an approach treats certain properties of the system model in particular ways, a simultaneously applied approach may undone these changes and thus adversely affect this approach.

For example, our *SecVolution* approach [1] evolves a information system according to explicitly modeled security knowledge. A precondition of a system model evolution is the compliance to the security knowledge (see Fig. 1). This means that the system model is analyzed to be secure according the security knowledge. SecVolution evolves the system to comply to changed security knowledge. On the one hand, other approaches are usually not aware of the relationship between system and security model. Thus, they change the system model without checking the security relations. On the other hand, other approach may evolve the system model in a way that it not compliant to the security knowledge. Since these approaches are not able to consider incompatibilities properly, the SecVolution approach needs to react on the evolutions. On that account, it has to re-establish the needed pre-requirements by analyzing conducted evolutions. Thus, this example shows that a common understanding and representation of evolution is required to react on evolution in a uniform manner.

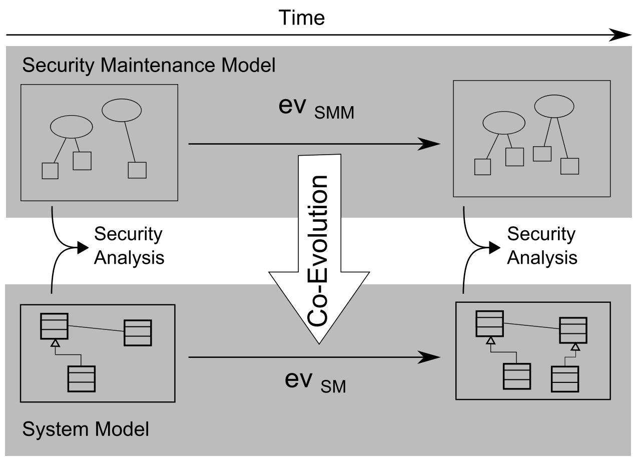

Fig. 1. Relationship between evolution and co-evolution as used in the SecVolution approach

The problem is that model changes are usually represented in form of fine-grained edit operations which are not suitable for re-establishing of the affected security model. In the area of security analysis, this often leads to a laborious and not fully automatic analysis of the model and, thus, requires evolution based strategies. While actual research in model-based security engineering [2], [3] provides approaches for hardening and maintaining the security of evolving systems, these approaches require semantic evolution information as input. In this paper, we define requirements on versioning and its interface to model-based security engineering of evolving systems. This will be illustrated by using the SecVolution, which is part of the priority programme "Design for Future: Managed Software Evolution"[4].

In his paper, the term *evolution* is defined as the ongoing change of development artifacts (e.g. software models, source code, natural-language documents) in a stepwise manner, such that every step preserves most properties (i.e. functionality and security) of the former system and is justified by a rationale. Here, evolution steps are defined as a transformation of models from their current state into a modified one. Additionally, we define co-evolution as an evolution of a target model such that a given relation between both models is preserved when the source model evolves.

Regarding the SecVolution approach, the relationship between evolution and co-evolution is depicted in Fig. 1. Here, we consider the evolution of security knowledge and the co-evolution of system models (SM) [5]. To measure the need for co-evolution, the system model is analyzed to be secure with respect to the modeled security knowledge. The security knowledge is modeled in the *security maintenance model (SMM)*. Whenever security knowledge evolves, the SMM is updated respectively, denoted by ev_{SMM}. In order to maintain the security of the system model, it also needs to evolve by applying ev_{SM}. Thus, the co-evolution for ev_{SMM} is the system model evolution ev_{SM}.

A major problem arises from the fact that it is necessary to exchange models between the ordinary development and the security maintenance processes using SecVolution. The pre-requirement of the co-evolution is, that the SMM and the system model are conform regarding the security analysis. Since most approaches evolving the system model are not handling security issues, we need react to on changes in the system model. Therefore, a common understanding of evolutions, their storage and their extraction out of modified models is necessary. For that purpose, we specify a set of essential requirements for evolutions in this paper. An important (if not the most important) request is to work with semantic evolutions which allows to gather differenced semantic changes scattered in the model.

The rest of this paper is structured as follows. To better understand the problem of a changed system model by a third-party approach, we describe the SecVolution approach in Sec. II. SecVolution provides the basis of characterizing the versioning and evolution requirements as presented in Sec. III. We conclude this paper in Sec. IV by focusing on ongoing research.

II. SecVolution - Beyond One-Shot Security

Evolution of information systems may be caused by changes in the environment including technical innovations, changing laws or security regulations, as well as evolving requirements. Compliance with increasingly ambitious security regulations requires information system updates which usually result in changes of various development artifacts. Thus, maintaining development artifacts of an information system efficiently while taking into account a continuously changing environment is a challenging task.

For supporting security requirements and design analysis for evolving information systems, we use the model-based development approach *SecVolution*. The overall goal of SecVolution is to restore security levels of an information system when changes in the environment put security at risk. As a core feature, our approach aims for reusing security-related knowledge and its evolution to facilitate co-evolution of development artifacts. During the evolution of a long-living information system, changes in the environment are therefore captured and translated to adaptations of the secure system model.

The SecVolution approach is built on the *SecReq* approach developed in our previous work [6], [7]. As a core feature, SecReq supports reusing security engineering experience gained during the development of security-critical software and feeding it back into the model-based development process. Therefore, SecReq combines three distinctive techniques to support security requirements elicitation as well as modeling and analysis of the corresponding system model: (1) Common Criteria [8] and its underlying security requirements elicitation and refinement process, (2) the HeRA tool [9] with its security-related heuristic rules, and (3) the UMLsec tool set [10] for secure system modeling and security analysis. This bridges the gap between security best practices and the lack of security experience among developers and designers. Once identified, a wizard guides analysts through a refinement process. As we argued before [6], [7], freeing valuable time of security experts is an essential benefit, because they can then deal with new threats that are not covered by automated reaction yet.

We store the security knowledge in the *security maintenance model* (SMM). It provides the security-relevant knowledge, including but not limited to security facts such as attacker types and their abilities, encryption protocols and their robustness against different attacks, etc. Moreover, the SMM contains additional adaptation information describing how the security-relevant knowledge can evolve and how to co-evolve a model of the secure information system that is built upon this knowledge, e.g. how to act if new attackers appear or if vulnerabilities change their conditions (see Fig. 1). Therefore, we incorporate ontologies [11] as an appropriate formalism flexible enough to model this information properly. Ontologies are further well established for representing knowledge in a formal and structured manner. Thus, it provides the foundation for computational inference required to determine co-evolution.

Figure 2 depicts the information flow of our approach. Input is derived from security-relevant documents such as guidelines or laws as well as from people such as white hats or system developers. Afterwards, retrieved security-relevant knowledge is formalized and encoded in the SMM. Each change of security-related knowledge is also formalized, so that the deltas can be used to adapt the secure system models to maintain the security of the information system they represent.

In SecVolution we suggest socio-technical methods for supporting elicitation of relevant changes. As presented in our previous work [12], [13], knowledge should be captured during the development task (not as a separate activity) with as little extra burden as possible for the expert and should be as little intrusive as possible to decrease effort for software engineers. Specific monitoring and elicitation techniques should therefore raise relevant information as a by-product to software engineering activities that are conducted anyway. Examples include a conversation between stakeholders, a demonstration of a new attack recently discovered by a security expert and manual changes of the system model performed by system developers. Specific techniques ought to be based on light-weight tool support covering different software development and maintenance stages. Tool support is intended to be optimized for minimal effort on the side of the security expert.

Changes trigger reactions, which lead to a co-evolution of security precautions and the corresponding secure system model. Updating security precautions is time-consuming and error-prone. For this reason, there should be automated reactions to the changes whenever possible. This requires formalization of changes addressed by the SMM. Given an originally valid system model, most of these adaptations can be directly derived from the environmental changes by means of co-evolution. For example, this includes predefined rules

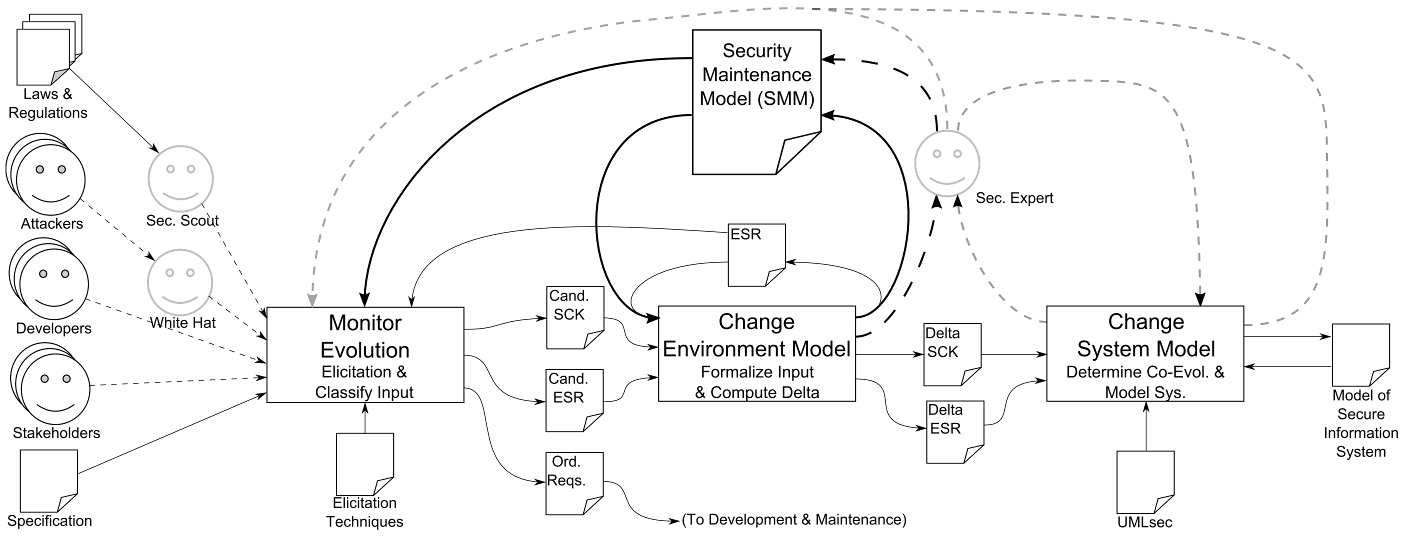

Fig. 2. Information flow model of the SecVolution approach (ESR = Essential Security Requirements; SCK = Security Context Knowledge).

describing how to change the secure system model in case of environmental changes. In SecVolution, we focus on UML models enriched by security-related annotations to express security requirements of a system model. For this purpose, we use the UML security extension UMLsec [10]. These annotations have to be updated to deal with the changed environment.

In summary, the SecVolution approach is intended to gradually wrap an existing information system into a layer of security-related knowledge, rationale, and proven solutions for known problems. The SMM provides the core security information and is connected to each activity as presented in Fig. 2. Greenfield development (i.e. development of new systems from scratch) is considered an exception which, however, can also be handled.

III. Requirements on model evolution

As seen in the SecVolution approach, information systems in model-based development consist of different software models such as system model and security model. Moreover, different development groups are involved in developing the system and at the most, they work on different parts of the system and of course have different viewpoints on the system. Every entity participating in the development process may cause changes to the corresponding models, without notifying the other participants. Thus, if there are commonly shared models among one information system, no development

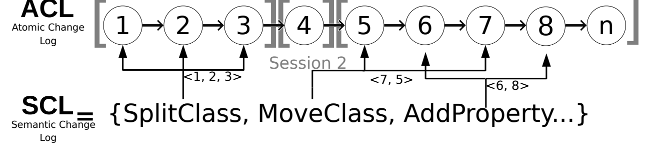

Fig. 3. Comparison of atomic change log and pattern change log: Atomic change operations can be presented by semantic change operations. The semantic changes need not to be continuously be present in the atomic changes.

participant can assume the models to be unchanged when trying to apply changes on its own. This means that every model-based approach used to develop software can change models and some other participants have to be informed about these changes. For a sophisticated reaction on evolution the semantic meaning is inevitable.

Two challenges arise from the purposes of extracting evolutions from edit operations: The first one arises by diversity of the change operations' granularity. Many tools allow predefined edit operations on models, but as edit steps can be of varying granularity, they do not necessarily describe evolution steps semantically. The second challenge is that different semantic evolutions can be performed in an interleaved manner.

For example, a system designer needs to change the system model of an information system twice. Each change is a separate and disjunct addition of functionality which is applied within the scope of a maintenance task and, thus, can be described semantically. However, each task may consist of several edit operations which modify the system model. Since both tasks induce changes in similar parts of the model, the designer changes some patterns (e.g. classes) for both tasks together, so that she has not to revisit this part later on. Thus, the designer interleaves the edit operations of both tasks: Different steps of the evolutions are executed in an arbitrary and mixed sequence. Figure 3 shows an illustration of our example to clarify this. Consider that the system designer performed three editing sessions. For each session, a sequence of atomic edit operations has been logged. These edit operations can resemble arbitrary semantic meaning (SplitClass, MoveClass and AddProperty). This means that we (wrt. Fig. 3) cannot simply regard a sequence of atomic operations as evolutions. Thus, it is desired to create a log of semantic changes based on pre-defined change patterns [14].

In summary, semantic evolutions define what the actual effect to the system is when the respective evolution is applied. In contrast to this are the atomic edit operations that resemble the respective evolution.

Software engineers who have to react on evolutions can benefit much from evolutions if they are represented on a semantic level. They do not need to extract the meaning of a change by themself. Especially, a huge amount of changes can be dealt with in a systematic, less laborious and less erroneous manner. Additionally, co-evolution of dependent models becomes even more applicable [15], [16].

For example, in SecVolution, whenever the system model is changed the security model needs to be co-evolved. A pragmatic way would be to re-create the security model, so that it naturally matches the model again. Since this is a laborious and expensive task, the existing security model should be adapted. The current approaches to do this involve manual modification. This is error-prone and the consistency with the model is not guaranteed. Figure 4 depicts this challenge of co-evolution: After an evolution of a system's model (ev_{Model}) has been applied or detected, it has to be investigated what is the appropriate evolution to be carried out on the security level (ev_{SecModel}) such that both the security model and the model of the system retain consistent afterwards.

Nowadays, models are serialized and stored using versioning systems which maintain their change history. The change history usually consists of fine-grained edit operations and does not contain information about the semantic effects of changes. Thus, we need a transparent and lightweight way to get the required information out of the version logs. For this purpose, we enumerate some essential requirements to a versioning system from the viewpoint of co-evolution. These requirements result from cooperations between different projects of the German Priority Program (SPP) "Design for Future - Managed Software Evolution" [17] (especially AdVERT [18], [19] and URES [20], [21]). All SPP projects work on different problems of evolution and many include changes in the system model.

Description language for semantic evolutions: Regarding Fig. 3, our goal is to abstract from atomic edit operations and describe evolutions on a semantic level. Since we assume that semantic evolutions can be expressed more generally, we call them change patterns [14]. The set of change patterns depends on the usage in different development techniques. Some change patterns are considered as semantic evolution in some but not all approaches. Thus, we require a modeling approach for representing semantic evolutions.

Semantic version logs: The version log of a versioning system should have a semantic view on the changes. Regarding Fig. 3, the log should contain information of the applied change patterns. This means that the differences between two versions of an artifact are described as a sequence of evolutions. Also the parameter values of the semantic evolutions should be explicitly given. For example, if a class is renamed, the new name should be explicitly given and not ,e.g., be hidden in an added code segment.

Completeness check of semantic evolutions: A given set of semantic evolutions may be insufficient to describe all changes contained in a version log. Therefore, there should be a check which validates that a set of semantic evolutions resembles all given changes.

Transparent and automatic computation: The above requirements should be realized transparently to all viewpoints.

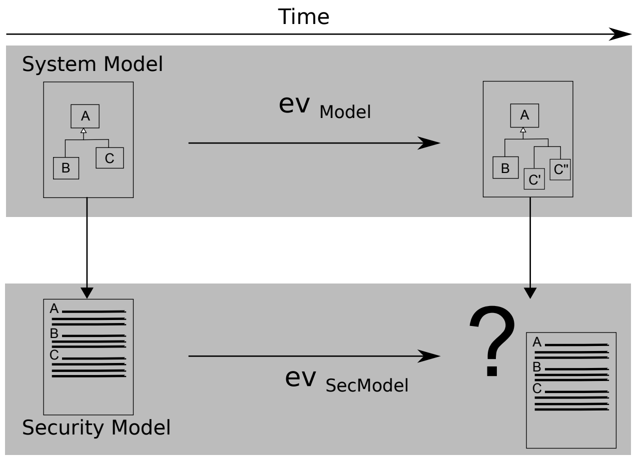

Fig. 4. Illustration of the challenge of co-evolution of a system's security model wrt. model evolution

From the users view he wants not to be interrupted in a development task for technical analyses of the changes he applied. Altogether the user should not notice the reinterpretation of his actions.

IV. Conclusion

The use of different development approaches with different viewpoints on software models results in the problem that models may be changed in a manner not suitable for some approaches. Nowadays, models are serialized and stored using versioning systems which maintain their change history. The change history usually consists of fine-grained edit operations and does not contain information about the semantic effects of changes. This means that the changes are given on a rather technical abstraction.

The need for semantic representation of changes has been discussed in the mentioned SecVolution project. It is a holistic approach for elicitation and maintenance of security-relevant requirements as well as the adaptation of the corresponding software models in case of changes. Here, security related parts of an information system have to be changed when requirements evolve or when assumptions about the context do not hold any longer. To retain a certain level of security of information systems, software engineers have to decide which changes of the software models must be applied. For this purpose, they need an understandable representation of changes.

For example, faster computers can make it easier to decode primitive cryptographic encodings. As a consequence, new approaches like rainbow tables [22] can compromise password encoding schemes that were considered sufficient before. A rainbow table is a precomputed table for reversing cryptographic hash functions. To protect the information system sufficiently, password policies should be improved. Thus, the associated security model of the information system must be co-evolved accoding to the new insights.

Using semantic evolution we will integrate different evolution and co-evolution approaches used in the SPP "Design for Future - Managed Software Evolution" in a compatible way.

After we have shown the need for semantic evolutions in contrast to atomic edit operations, we listed necessary requirements to the next generation of versioning systems. A major challenge is to develop a system which transparently provides both viewpoints and can generate the missing one out of the existing one.

Acknowledgment

This research is funded by the DFG project SecVolution (JU 2734/2-1 and SCHN 1072/4-1) which is part of the priority programme SPP 1593 “Design For Future - Managed Software Evolution”.

References

[1] J. Jürjens and K. Schneider, “Beyond one-shot security,” in *Modelling and Quality in Requirements Engineering (Essays Dedicated to Martin Glinz on the Occasion of His 60th Birthday)*. Verlagshaus Monsenstein und Vannerdat, 2012, pp. 131–141.

[2] J. Jürjens, “UMLsec: Extending UML for secure systems development,” in *Proc. of the 5th International Conference on the Unified Modeling Language (UML)*, ser. LNCS, vol. 2460. Springer, 2002, pp. 412–425, 10 years most influential paper award at Models 2012.

[3] T. Ruhroth and J. Jürjens, “Supporting Security Assurance in the Context of Evolution: Modular Modeling and Analysis with UMLsec,” in *Proc. of the 14th International Symposium on High-Assurance Systems Engineering (HASE)*. IEEE, October 2012.

[4] Homepage of SPP, “Design for future: managed software evolution.” [Online]. Available: http://www.dfg-spp1593.de

[5] T. Ruhroth and H. Wehrheim, “Model evolution and refinement,” *Science of Computer Programming*, vol. 77, no. 3, pp. 270 – 289, 2012.

[6] S. H. Houmb, S. Islam, E. Knauss, J. Jürjens, and K. Schneider, “Eliciting security requirements and tracing them to design: an integration of Common Criteria, heuristics, and UMLsec,” *Requirements Engineering*, vol. 15, no. 1, pp. 63–93, 2009.

[7] K. Schneider, E. Knauss, S. Houmb, S. Islam, and J. Jürjens, “Enhancing security requirements engineering by organizational learning,” *Requirements Engineering*, vol. 17, no. 1, pp. 35–56, 2011.

[8] International Standardization Organization, “ISO 15408:2007 Common Criteria for Information Technology Security Evaluation, Version 3.1, Revision 2, CCMB-2007-09-001, CCMB-2007-09-002 and CCMB-2007-09-003,” International Organization for Standardization (ISO), 2007.

[9] E. Knauss, D. Lübke, and S. Meyer, “Feedback-Driven Requirements Engineering: The Heuristic Requirements Assistant,” in *Proc. of the 31st International Conference on Software Engineering (ICSE)*, 2009, pp. 587 – 590.

[10] J. Jürjens, *Secure Systems Development with UML*. Springer, 2005.

[11] J. F. Sowa, *Knowledge representation: logical, philosophical, and computational foundations*. MIT Press, 2000, vol. 13.

[12] K. Schneider, *Experience and Knowledge Management in Software Engineering*. Springer, 2009.

[13] ——, *Rationale Management in Software Engineering*. Berlin, Heidelberg: Springer, 2006, ch. Rationale as a By-Product, pp. 91–109.

[14] T. Kehrer, U. Kelter, M. Ohrndorf, and T. Sollbach, “Understanding model evolution through semantically lifting model differences with SiLift,” in *Proc. of the 28th IEEE International Conference on Software Maintenance (ICSM)*, 2012, pp. 638 –641.

[15] T. Ruhroth and H. Wehrheim, “Refinement-Preserving Co-evolution,” in *Proc. of the 11th International Conference on Formal Engineering Methods and Software Engineering (ICFEM)*, K. Breitman and A. Cavalcanti, Eds., vol. 5885. Springer, 2009, pp. 620–638.

[16] J. Jürjens, “Automated security hardening for evolving uml models,” in *Proc. of the 33rd International Conference on Software Engineering (ICSE)*. ACM, 2011.

[17] SPP 1593 - ”Design for Future - Managed Software Evolution, “Homepage,” 2014. [Online]. Available: http://www.dfg-spp1593.de/

[18] Z. Durdik and R. Reussner, “On the Appropriate Rationale for Using Design Patterns and Pattern Documentation,” in *Proc. of the 9th ACM SIGSOFT International Conference on the Quality of Software Architectures (QoSA)*, 2013.

[19] M. Konersmann, Z. Durdik, M. Goedicke, and R. Reussner, “Towards architecture-centric evolution of long-living systems (the ADVERT approach),” in *Proc. of the 9th ACM SIGSOFT International Conference on the Quality of Software Architectures (QoSA)*, 2013, pp. 163–168.

[20] T. Roehm, B. Bruegge, T.-M. Hesse, and B. Paech, “Towards identification of software improvements and specification updates by comparing monitored and specified end-user behavior,” in *Proc. of the 29th International Conference on Software Maintenance (ICSM)*. IEEE, 2013, pp. 464–467.

[21] T.-M. Hesse and B. Paech, “Supporting the collaborative development of requirements and architecture documentation,” in *Proc. of the 3rd International Workshop on the Twin Peaks of Requirements and Architecture*. IEEE, 2013, pp. 22 – 26.

[22] P. Oechslin, “Making a Faster Cryptanalytic Time-Memory Trade-Off,” in *Advances in Cryptology - CRYPTO 2003*, ser. LNCS, D. Boneh, Ed. Springer, 2003, vol. 2729, pp. 617–630.