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.ruhrroth,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—Information systems are exposed to constantly changing environments which require constant updating. Moreover, security is an increasingly important quality facet in modern information systems and needs to be retained. Thus, long-living software systems “ages” not by wearing out, but by failing to keep up-to-date with its environment. 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, an evolution strategy of the associated security model can be chosen. 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

Degradation of security can be a serious problem for long-living information systems due to environmental changes including technical innovations, changing laws or security regulations, as well as evolving requirements. This especially applies to information systems which process critical data, and thus security has to be particularly considered when maintaining such systems. To ensure security of information systems, technical solutions are primarily established and evaluated by security experts. Therefore, it is necessary to determine the impact of changes on relevant security properties [1] of the regarded system. Otherwise the system can erode if its environment changes. This is a critical situation especially if the cause of a change is directly induced by security reasons, e.g. by newly revealed exploits or new attacking capabilities. Then, one has to know which parts of the information system are affected and have to be adapted.

In model-based development, the information system is developed by means of models, e.g. system model, security model, etc. Accordingly, evolutionary changes of the regarded system model have to be understood by software engineers in order to establish or adapt security measures represented as security model. The problem is that model changes are usually represented in form of fine-grained edit operations which are not suitable for choosing appropriate evolution strategies of the affected security model. 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 an input. In this paper, we define some requirements on versioning and its interface to model-based security engineering of evolving systems. This will be illustrated by using the SecVolution project [4], which is part of the priority programme “Design for Future: Managed Software Evolution”[5].

The SecVolution approach is based on the notion of evolution and co-evolution. The term evolution can be defined as the ongoing change of development artifacts in a stepwise manner, such that every step preserves most properties (functionality and security) of the former system and is justified by a rationale. This evolution steps can be seen as a transformation of models from their current state into a modified one.

Using an evolution transformation which formalizes the evolution of a model, we can define a co-evolution as an evolution of a second model such that a given relation between both models is preserved when the first 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) [6]. The security knowledge
is modeled in the security maintenance model (SMM). To measure the need for co-evolution, the system model is analyzed to be secure with respect to the modeled security knowledge.

Now, the SMM changes whenever new security knowledge is captured. The evolution is depicted as $\text{ev}_{\text{SMM}}$. In order to maintain the security of the system model, it also needs to evolve by applying $\text{ev}_{\text{SM}}$. Thus the co-evolution for $\text{ev}_{\text{SMM}}$ is the system model evolution $\text{ev}_{\text{SM}}$.

A major problem arises from the fact that we need to exchange models between the ordinary development and the security maintenance processes using SecVolution. Therefore, a common understanding of evolutions, their storage and their extraction out of modified models is much-needed.

The rest of this paper is structured as follows: In Sec. II, we present the SecVolution project. 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 propose 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 [7], [8]. 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 [9] and its underlying security requirements elicitation and refinement process, (2) the HeRA tool [10] with its security-related heuristic rules, and (3) the UMLsec tool set [11] 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 [7], [8], 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.

The core of SecVolution is 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. Therefore we incorporate ontologies [12] 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 [13], [14], 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 lightweight 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 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 [11]. These annotations have to be updated to deal with the changed environment.

In summary, the SecVolution approach is intended to grad-
ually 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 ARTIFACT EVOLUTION

Information systems mostly consist of different development artifacts like models, analysis documents or code. In general, several 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 various artifacts without notifying the other participants. Thus, if there are commonly shared development artifacts among one information system, no development participant can assume the artifacts to be unchanged when trying to apply changes on its own, i.e. evolutions.

Two challenges arise from the purposes of extracting evolutions from edit operations: One challenge arises by diversity of the change operations’ granularity. Many tools allow arbitrary edit operations on arbitrary artifacts, but as edit steps can be of varying detail-levels, they do not necessarily describe evolution steps in terms of a semantic entity.

Consider the following example: a system designer implements two changes (i.e. evolutions) in the model of an information system. While the two evolutions are two individual evolution steps that are first defined on a semantic level, the implementation comes down to a number of edit operations in model views. Moreover, the designer may interleave the edit operations of both the evolutions. This may at least result from convenience reasons because the designer may first do all necessary editing in the first model and edit a second model afterwards. Modifications to artifacts like text files mainly result in fine granularity edit operations, namely addition and removal of model elements, identifiers, etc. Figure 3 shows an illustration of our example to clarify this. Consider that the developer did three editing sessions to implement two changes. For each session, a sequence of atomic edit operations has been logged. Since these edit operations can resemble arbitrary semantic results, it is difficult to recognize the semantic effect of the operations. For a sophisticated reaction on evolution the semantic effect is inevitable. Thus, it is necessary to extract the semantics of evolutions and create a log of changes at the level of change patterns [15].

The second challenge is, as mentioned above, that different semantic evolutions can be performed in an interleaved manner. This means that we (wrt. Fig. 3) cannot simply regard a sequence of atomic operations as evolutions.

Nonetheless, projects that react on evolutions can benefit much from evolutions if they are represented on a semantic
level. For example, a common problem is to co-evolve specialized artifacts, whenever the model of an information system has evolved [16], [17].

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

Semantic evolutions are an abstraction from simple edit operations and 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. One example is moving a full class vs. edit operations in an artifact file. Following this example, the edit operations regarding a Java project could be creating a new file, copying the contents of one class file to another, renaming the references and finally deleting the old source file. Each step itself represents an evolution but cannot as easily treated as an evolution “move class” covering all the respective edit steps.

Most artifacts are stored in versioning systems, but normally the change logs do not consist of semantic evolutions. Thus, we need a transparent and lightweight way to get the required information out of the version-logs.

As a next step, we enumerate some requirements to a versioning system from the viewpoint of co-evolution.

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. 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 applied change patterns. This means, that the differences between two versions of an artifact are described as a sequence of evolutions. Also additional information for these evolutions should be provided. For example, if a class is renamed, the new name is needed while performing the evolution.

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. 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 techniques with different viewpoints on project artifacts results in the problem that artifacts may be changed in a manner not suitable for some techniques. Nowadays, many persistent artifacts are stored using versioning systems which maintain the change history of the artifact. This history information usually consists of fine-grained edit operations and does not contain information about the semantic effects of changes.

One example for the need of semantic evolutions is the mentioned SecVolution project. SecVolution is a holistic approach for elicitation of security-relevant requirements, maintenance of these requirements, and adaption of the realizing software in case of changes. It uses semantic evolution and co-evolution to reduce the workload of the laborious task of maintaining information systems carrying many security requirements. Security related parts of an information system have to be changed when requirements change or when assumptions about the context do not hold any longer. For example, faster computers can make it easier to decode primitive cryptographic encodings. As a consequence, new approaches like rainbow tables [18] can compromise password encoding schemes that were considered sufficient before. A rainbow table is a precomputed table for reversing cryptographic hash functions. If a system is adapted because of new security knowledge such as how password checks should be improved, this adaption could also be applied to other systems that might suffer the same problem.

While evolutions formalize changes to the security-relevant knowledge, co-evolutions are used to trigger necessary changes to the regarded system model. For this purpose, model transformations are used since the evolution and co-evolution rules can be declared well for modeling languages.

After we have shown the need of semantic evolutions in contrast to atomic edit operations, we listed necessary require-
ments to the next generation of versioning systems. Current versioning approaches are often code or atomic edit based. This means that the changes are given on a rather technical abstraction. But as presented in our SecVolution approach more advanced techniques need to be based on semantic evolutions. The requirements on versioning systems describe the need of having both viewpoints on a change log in parallel. A major challenge is to find 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


