Supporting Security Assurance in the Context of Evolution: Modular Modeling and Analysis with UMLsec

Thomas Ruhroth
Technische Universität Dortmund
thomas.ruhroth@cs.tu-dortmund.de

Jan Jürjens
TU Dortmund & Fraunhofer ISST
http://jan.jurjens.de

***Abstract*—Developing security-critical software correctly and securely is difficult. To address this problem, there has been a significant amount of work over the last 10 years on providing model-based development approaches based on the Unified Modeling Language which aim to raise the trustworthiness of security-critical systems. However, the fact that software continues to evolve on an ongoing basis, even after the implementation has been shipped to the customer, increases the challenge since in principle, the software has to be re-verified after each modification, requiring significant efforts. In particular, as part of the system evolution, the threat model can change against which the design has to be verified. This requires a modular approach to security assurance, since the threat model has to be substituted independently from the design model. In this paper, we present such an approach based on the extension mechanisms available for the Unified Modeling Language (UML), in particular using so-called profiles. This modular approach allows us to define analysis models which can be exchanged easily whenever the threat model changes due to system evolution. We demonstrate the approach in the face of a specific security requirement, namely secure information flow.**

I. Introduction

Software-based systems are becoming increasingly long-living [1]. This was demonstrated strikingly with the occurrence of the year 2000 bug, which occurred because software had been in use for far longer than its expected lifespan. Also, software-based systems are getting increasingly security-critical since software now pervades the whole critical infrastructures dealing with critical data of both nations and also private individuals. There is therefore a growing demand for more assurance and verifiable secure IT systems both during development and at deployment time, in particular also for long living systems. Yet a long lived system also needs to be flexible, to adapt to evolving requirements, usage, and attack models. However, using today's system engineering techniques we are forced to trade flexibility for assurance or vice versa: we know today how to provide security or flexibility taken in isolation. We can use full-fledged verification for providing a high-level of assurance to static requirements, or we can provide flexible software, developed in weeks using agile methodologies, but without any assurance. This raises the research challenge of whether and how we can provide some level of security assurance for something that is going to change.

Our objective is thus to develop techniques and tools that ensure "lifelong" compliance to evolving security requirements for a long-running evolving software system. This is challenging because these requirements are not necessarily preserved by system evolution [2]. In this paper, we present results towards a security assurance approach which can deal with evolution and in particular changing threat models. Most existing assessment methods used in the development of secure systems are mainly targeted at analyzing a static picture of the system or infrastructure in question. For example, the system as it is at the moment, or the system as it will be when we have updated certain parts according to some specifications. In the development of secure systems for longevity, we also need assurance approaches that can deal with changing threat models. Consequently, our approach allows interchanging the used analysis model when evolution takes place. This is done by making use of the extension mechanisms available for the Unified Modeling Language (UML).

Evolution describes the changes of a system in the maintenance phase of its lifecycle. Often evolution is driven by the change of requirements or by changes in the surrounding environment. Beside the change request by the system users, changes in security needs or changes in industrial standards like certification standards are very common triggers for evolution.

In model based development those security or standard requirements and properties are often denoted in the models. So a system architect or a developer can analyze the software early in the development process. One technique for model based security modeling is provided by UMLsec [3]. UMLsec annotates UML diagrams with security information using stereotypes. Also UMLsec provides so called *checks* to ensure the annotated properties.

Unfortunately, most approaches for UML are not prepared for changes in background definitions for checks. For example, whenever a security standard is changed, the old version of UMLsec needed to be adapted to comply the new standards.

Many of these problems are caused by two types of extensions provided in UML: the lightweight and the heavyweight extensions. Both types have their drawbacks in supporting evolution.

The lightweight extension uses stereotypes to annotate UML elements. Stereotypes are often useful but do not give the full flexibility to hold all information in the model. The solution provided by UML is to use a heavyweight extension by creating a new or extended meta-model. While giving full flexibility in meta-modeling, the heavyweight extension is not supported by most existing tools. Thus a full tool chain for the model based development approach has to be created.

In this paper we develop a simple and modular approach to combine the benefits of both extension types. As running example we use two new profiles for UMLsec. This profiles extend UMLsec with a notation and analysis of for secure information flow (SIF) using the *modular assembly kit for security* [4]. Since UMLsec already has support for other SIF techniques, the exchangeable UMLsec profiles give a sound technique for evolve UMLsec. It does not invalidate old models and give the possibility to use other notions of SIF. To archive the interchange of analysis models we defined a new kind of profiles by combining both kind of extension. The new kind of profiles have a syntax part, defined by a lightweight extension and an *analysis model* defined by an heavyweight extension.

The analysis model is a special meta-model for the analysis that is not directly represented in the genuine system model. It is used additionally to the lightweight extension to UML and offers many benefits from a heavyweight extension of UML in a lightweight way. The analysis model, seen as a heavyweight extension, allows extending the UML meta-model. The meta-model and the analysis models can be merged into a resulting analysis model.

After giving a brief introduction to the domain of the example application and UMLsec, in the third section we describe the structure of our extension. Then we demonstrate the approach by defining two UMLsec 2.0 subprofiles.

II. Model-based Security Assurance with UMLsec

UMLsec: Model-based Security Engineering (MBSE, [3], [5]) is a soundly based approach for developing security-critical software where recurring security requirements (such as secrecy, integrity, authenticity) and security assumptions on the system environment can be specified either within a UML specification or within the source code as annotations (cf. Fig. 1). Analysis plugins in the associated UMLsec tool framework [6] (Fig. 2) generate logical formulas formalizing the execution semantics and the annotated security requirements. Automated theorem provers (ATPs) and model checkers automatically establish whether the security requirements hold. If not, a Prolog-based tool automatically generates an attack sequence violating the security requirement, which

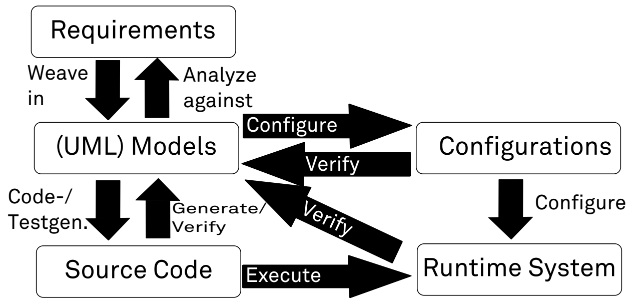


Figure 1. Model-based Security Engineering

can be examined to determine and remove the weakness. The UMLsec extension is given in form of a UML profile using the standard UML extension mechanisms. *Stereotypes* are used together with *tags* to formulate the security requirements and assumptions. *Constraints* give criteria that determine whether the requirements are met by the system design by referring to a precise semantics of the used UML fragment defined in [3] using so-called *UML Machines*, which is a kind of state machine with input/output interfaces and UML-type communication mechanisms. Advanced users of the UMLsec approach can also implement plugins for constraints of self-defined stereotypes. The UMLsec approach has been used in several industrial applications (cf. [7] and references there).

As UMLsec incorporates many security areas, in UMLsec the problem of exchanging and combining security approaches described in the introduction occurs.

In Fig. 3 a simplified part of a WebPaySystem component is depicted. A `Customer` can get an invoice and pay this invoice via a credit card. These credit card information can be stored. A misbehavior of a customer can be punished. All these actions are reflected in the operations of the class. The customer provides two interfaces. The interface `SellerInterface` exports the invoice operation and the

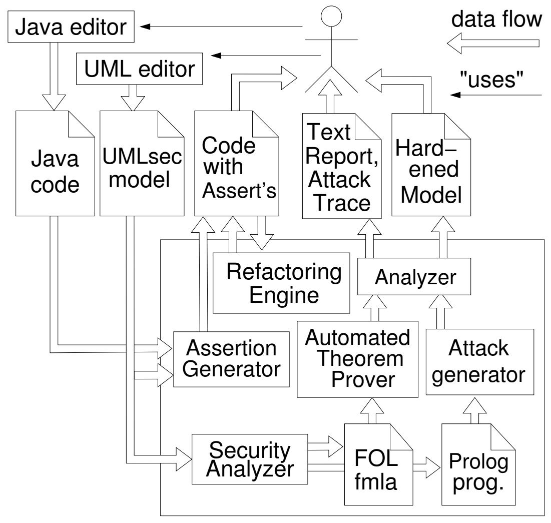


Figure 2. Model-based Security Tool Suite

`CustomerInterface` provides the operations to store the credit card information and to pay.

The state chart specifies that if an invoice has been issued, the customer has to pay this invoice before getting another invoice.

The idea of security modeling using UMLsec is to add security information to design and implementation models. A stereotype is an annotation in UML which can be parameterized by *tags* (see Fig. 3). The interface `Customerinterface` has two stereotypes. The stereotype ⟨⟨interface⟩⟩ is a UML stereotype annotation indication that this is an interface. The second stereotype ⟨⟨SecurityLevel⟩⟩ is a UMLsec SL stereotype parametrized by a tag named $level$ which is set to a set containing one element ($customer$).

While the syntax has nearly not changed in the new version of UMLsec, the system of the analysis has changed rapidly through the introduction of the analysis model and the new UMLsec 2.0 profiles. The latter parts are explained in this paper.

The analysis can be automatically performed for many security properties by the tool CARiSMA [8] or its predecessor UMLsecTool [6].

MAKS: The *Modular Assembly Kit for Security* (MAKS) [4], [9] can be used to express secure information flow properties in a modular and uniform way. The fundamental elements are *Basic Security Predicates* (BSP). All BSPs are parameterized with a view $\mathcal{V}$, denoting the elements which are confidential. A MAKS *view* $\mathcal{V} = (V, N, C)$ [4] is a disjoint partition of the event set E into three sets V, N, C.

	visible	invisible
confidential	$\emptyset$	C
not confidential	V	N

The set C collects confidential events which should not be seen. The events of V are visible and hold non confidential data. Events which cannot be observed and are not confidential are collected in the set N. The combination of visible but confidential events makes no sense, because visibility and confidentiality are contradicting.

The system model is given as a prefix closed set Tr of all system traces where a trace is a sequence of events E. The system traces Tr are prefix closed if for all traces also all prefixes are in Tr. For example, if the sequence $\langle abc \rangle$ is in Tr then also the traces $\langle ab \rangle$, $\langle a \rangle$ and $\langle \rangle$ need to be in Tr.

Using a view $\mathcal{V}$ we can define BSPs. One example for such a BSP is *strict removal* (SR):

$$SR_{V,N,C}(Tr) := \forall \tau \in Tr : \tau|_{V \cup N} \in Tr$$

Strict removal describes that all "confidential" events C are independent of the "visible" V and "neither-nor" events N and thus no information about confidential events can be inferred from the others.

BSPs can be combined and be parameterized by views. By using BSPs many SIF properties can be expressed, including many traditional notions. For example non-interference ($NF_{H,L}(Tr) := \forall \tau \in Tr : \tau|_L \in Tr$) can be modeled using SR [4]:

Let L be a set of low events and H be a set of high events with $L \cup H = E$. Then the following equality holds:

$$SR_{H,\emptyset,L}(Tr) = NF_{H,L}(Tr)$$

Security Levels and Posets (Partially Ordered Sets): Many complex systems define security levels to describe which data can be seen by a user and which not. A widely known system is the linear system of the NATO defining the security levels unclassified, classified, secret and top secret. In general a security level structure is described by a *partially ordered set* $(A, \leq)$.

If P denotes the set of all permissions, then a security level can be described as a subset of P. The family of security levels together with the operation subset $\subset$ builds a poset. This can be depicted as a Hasse Diagram (see Fig.9). Each connection from a higher x to a lower y node means $x \leq y$.

In this paper the security levels are modeled as sets of events that are allowed to be seen by users holding this permission. Therefore the relation between the security levels is modeled as a poset.

III. Security Assurance in the Context of Evolution: Exchangeable Profiles

As discussed in the introduction, our goal is to use the benefits of heavyweight UML extensions but not to loose the flexibility of lightweight extensions. As a side-condition we want to minimize the changes of the UMLsec syntax. Thus we define a new kind of profiles consisting of a lightweight extension defining the syntax and heavyweight extension corresponding to the semantics.

The idea of the lightweight extension is to support the addition of information to a model while the definition of UML, the meta-model, is neither changed nor parts are removed. The technique for adding information is to use stereotypes. They do only add information that can be ignored by tools not knowing a stereotype.

On one hand lightweight extensions are well supported by tools but is not expressive enough to easily define the analysis. On the other hand heavyweight extensions defining a new meta-model and thus has the full power in changing the UML meta-model, but is complicated to combine with existing tool chains.

Our new profiles combine the advantages of both the lightweight and the heavyweight extension (Fig. 4). Each analysis profile consists out of a normal lightweight UML profile defining stereotypes, an analysis model defining a

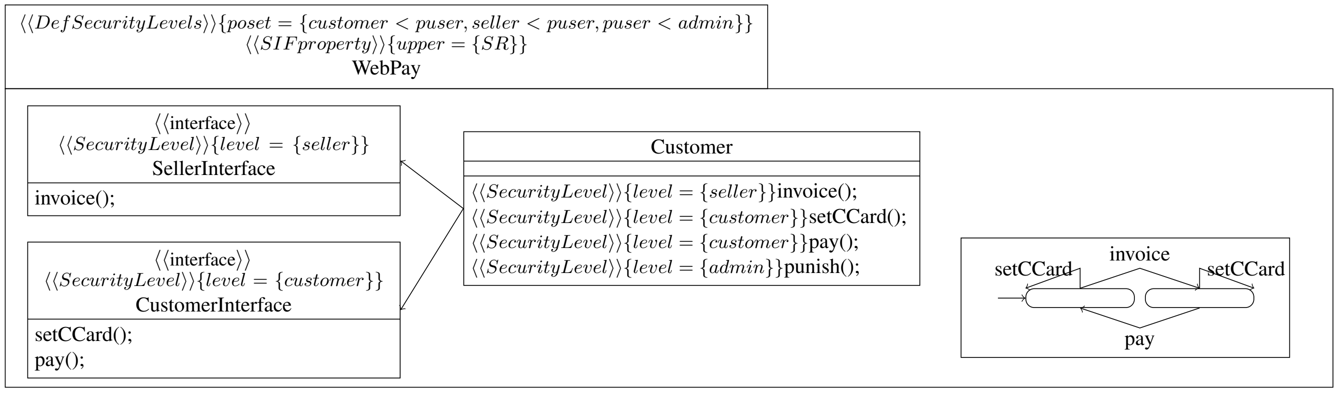

$\langle\langle DefSecurityLevels\rangle\rangle\{poset = \{customer < puser, seller < puser, puser < admin\}\}$
$\langle\langle SIFproperty\rangle\rangle\{upper = \{SR\}\}$
WebPay
⟨⟨interface⟩⟩
$\langle\langle SecurityLevel\rangle\rangle\{level = \{seller\}\}$
SellerInterface
invoice();
⟨⟨interface⟩⟩
$\langle\langle SecurityLevel\rangle\rangle\{level = \{customer\}\}$
CustomerInterface
setCCard();
pay();
Customer
$\langle\langle SecurityLevel\rangle\rangle\{level = \{seller\}\}$invoice();
$\langle\langle SecurityLevel\rangle\rangle\{level = \{customer\}\}$setCCard();
$\langle\langle SecurityLevel\rangle\rangle\{level = \{customer\}\}$pay();
$\langle\langle SecurityLevel\rangle\rangle\{level = \{admin\}\}$punish();
setCCard
invoice
setCCard
pay

Figure 3. Simple Example of a Web-Payment System using UMLsec

heavyweight extension and a mapping between them. The stereotype profile is used for modeling and can be used in tools supporting custom profiles. For complex analysis the analysis model is used. The analysis model is often an extension of the basic UML-meta-model. In the analysis model are data structures for the analysis defined and constraints are be given using OCL. The profile is completed by a transformation describing how the annotations given by stereotypes can be transformed in an analysis model.

The separation supports the exchange of profiles, so the analysis model can be exchanged without changing the syntax. In some cases are modifications restricted, e.g., the SL profile can only be exchanged by a profile including the parts used by the SIF analysis.

In the next section we demonstrate the use of the new combined profiles in the new version of UMLsec called UMLsec 2.0. Figure 4 gives an overview over the profiles and the approach: Each profile is constructed out of three parts. The first one is a standard UML profile package defining stereotypes. The stereotypes can be used to annotate UML model elements. UML profiles are the basis for the storage of the information. The annotated models can be saved and processes with the normally used tools, e.g., by using the XMI exchange format.

Other parts are used for an analysis and are only used in the parts performing the analysis. The analysis model defines a new meta-model defining the structure used for an analysis. We call it "analysis model" since we are interested in the analysis of model properties, but it can be used for arbitrary purposes. To fill the analysis model with data a transformation is applied generating the object structure out of the model annotated using stereotypes. This transformation can directly map the stereotypes to objects (for example used in the UMLsec 2.0 SIF profile) or it can have a stepwise transformation like it is used latter in the SL profile. Both kinds lead to an object structure used for the analysis. The preferred way for the analysis is to define checks directly on the analysis model using OCL, but the approach is not limited to OCL.

A UMLsec profile can use other profiles. We can aggregate commonly used parts in a profile which can be reused. For example, the security level profile is used in arbitrary contexts like secure information flow, security classification of documents or usage control. The composition of UMLsec profiles is done for each part of the profile separately. The stereotype names have to be disjoint. Thus all stereotypes of the profiles can be used in combination. The analysis models are merged using the merge operation defined in the UML standard. The instantiation of the model follows the include relation between the profiles. A transformation into the analysis model is performed when all used analysis models are instantiated.

The analysis is triggered by the top profiles, which are the profiles not used by other profiles. The analysis of used profiles is not performed by itself. If such an analysis is needed it must be triggered by the top profile as part of its own analysis.

Now, having introduced all parts we present this approach using the new UMLsec profile for the analysis of secure information flow using MAKS. This analysis uses the security level profile.

IV. Exchangeable Analysis Models: Modeling Secure Information Flow

In this section we explain the proposed profile technique while applying it to UMLsec.

Secure information flow is a good example for applying of the proposed profile technique: There are different approaches formalizing SIF properties. Often the different definitions are conflicting, so in the old UMLsec definition one approach was chosen to be included. Thus, the old version of UMLsec only supports the Bell–LaPadula model [10].

Since security levels are used in in different SIF definitions as well as other security properties, it is convenient to model the security levels on their own.

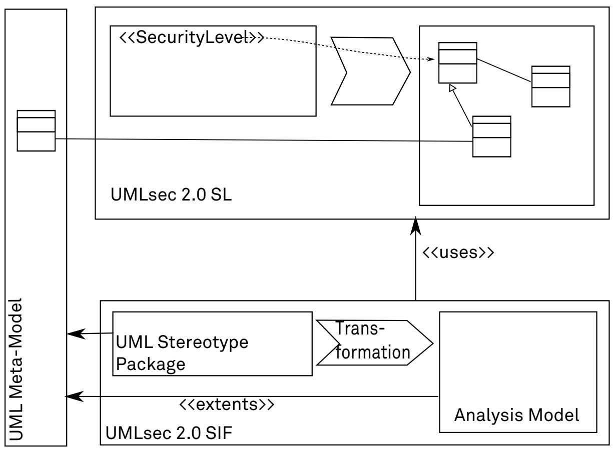


Figure 4. Structure of a UMLsec-Profile

Therefore, we present two UMLsec 2.0 profiles: The SL profile defines the security levels and the SIF profile defines secure information flow using MAKS. Each profile defines some stereotypes used in the modeling as well as an analysis model used for the analysis. Because the SIF profile uses the SL profile, we start with the latter.

UMLsec 2.0 SL: Defining security levels usable in other sub-profiles is the main task of the UMLsec 2.0 SL profile. We start describing the syntax defined in the SL profile by giving the stereotypes.

The SL profiles defines two stereotypes (see Fig. 7). ⟨⟨DefSecurityLevels⟩⟩ is used to define the security levels. The actual definition is given in the tag {`levelPoset`}. The poset is given by a collection of relations. All possible security levels are given by collecting the security level names given in the relations. The poset is defined as the transitive closure of the given relations. In Fig. 3 for the package `Webpay` are security levels defined. The defined poset has the elements “customer”, “puser”, “seller” and “admin”.

The second stereotype ⟨⟨SecurityLevel⟩⟩ assignes a specific security level to an element of an UML diagram. In the example the security level *customer* is assigned to the class `CustomerInterface`.

Many approaches using security level do not work on the defined levels itself. In the area of security analysis techniques often a small set of two or three defined security levels (here we will call them *basic security levels* or BSL) are used. Often two basic security levels are defined which are called *high* for confidential or secure data and *low* for unclassified data. Thus we need to map generic security levels to the basic security levels used for an analysis. At the same time we want to retain the connection of the calculated basic security levels to the original security levels and the annotated model elements. As result we can track back problems found in the latter analysis back to the annotated model.

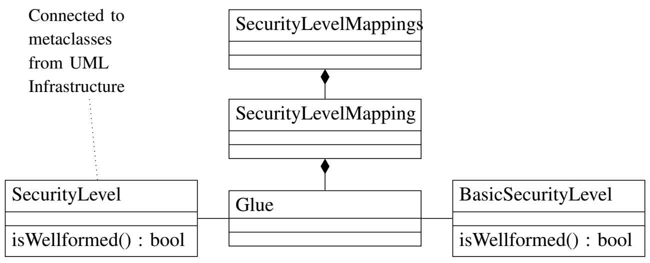


Figure 5. Analysis Model of UMLsec 2.0 SL Profile

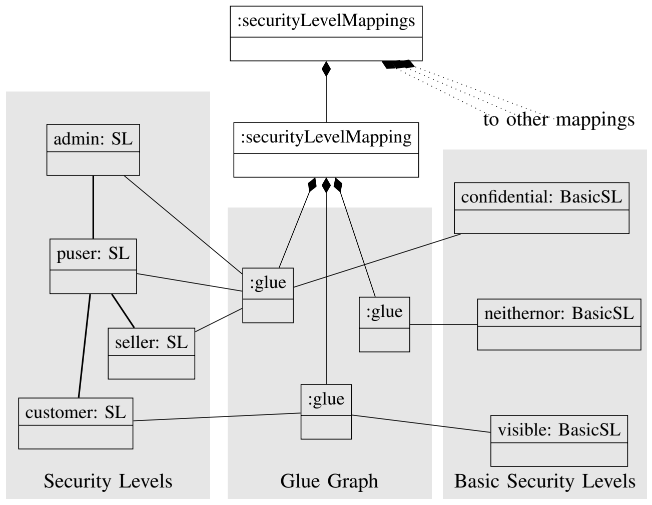


Figure 6. Example of using UMLsec 2.0 SL Analysis Model. One mapping is shown explicitly; the others are hidden in the dotted composite associations. The connections to the meta classes from the UML Infrastructure are omitted.

Since some approaches like MAKS use more than two BSLs (e.g. MAKS: “confidential”, “neither-nor“ and ”visible“) the system in our analysis model is modeled in a flexible way.

The SL analysis model consists out of three columns (see Fig. 5 and 6) is using techniques from triple graph grammars [11]. On the right the basic security levels are modeled. The framework for arbitrary security levels is defined on the left side and both columns are connected using glue graphs.

The instances of the class `SecurityLevel` are the security levels which are annotated in the UML model. The security levels form a hierarchy, which is modeled using the super relation. The hierarchy is sound if it does not contain circles, which ensures that the hierarchy builds a poset.

To illustrate the SIF modeling, we transform the example Fig. 3 which is depicted in Fig. 6: On the right side are the objects representing the values needed by the security properties to be checked. Since we use MAKS to analyze the secure information flow, we have three basic security levels representing the MAKS view: “confidential”, “neither-nor” and “visible”. On the left side there are instances for the

Stereotype	Tags	Use with
Package SL (Security Level)		
SecurityLevel	level	Operation, Property Class, Interface
DefSecurityLevels	levelPoset	Package, (Sub-)System Component
Package SIF (Secure Information Flow)		
SIFproperty	upper, lower	Package, (Sub-)System, Component

Figure 7. Stereotypes

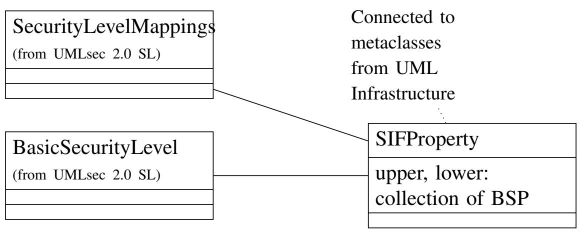


Figure 8. Analysis Model of UMLsec 2.0 SIF Profile

security levels. In the middle a glue graph is connecting the two layers. Each glue graph element maps all security levels to the view sets. The class `SecurityLevelMappings` collects the valid mappings. Each valid mapping is modeled by a `SecurityLevelMapping` where the glue objects collect the `SecurityLevels` and establish the mapping to the `BasicSecurityLevels`. Each of the glue objects can be interpreted as one instance of the views needed to analyze the poset on the left. Information about the derivation of the mappings is given in the section about the MAKS analysis.

UMLsec 2.0 SIF: Different secure information flow properties should be to usable in UMLsec 2.0. For every SIF notion we need a separate profile. Here we present the analysis using the *Modular Assembly Kit for Security* (MAKS) [4], [9]. The SIF properties are defined by BSPs. Since these BSPs are well-defined, we use their names to define the SIF properties of a system. Thus we define a stereotype ⟨⟨SIFproperty⟩⟩ with tags “upper” and “lower”. Each tag takes a collection of BSPs.

The SIF analysis model uses the SL analysis model, in Fig. 8 on the left side the classes `BasicSecurityLevel` and `SecurityLevelMappings` are imported from SL. The new class `SIFproperty` has a reference to the defining UML element denoting the scope of the SIF property. The values of the tags are stored in instance variables.

Up to here we defined the annotations of the model and their analysis models. In the next step we use the analysis models to check the security properties.

V. Exchangeable Analysis Models: Analysis of Secure Information Flow

In the last section we focused on the modeling of secure information flow related data and properties. With the data of the analysis model we can define an analysis which can use all data of the heavyweight extension’s meta-model. Each transformation (`securityLevelMapping` in Fig.6) in the analysis model leads to a MAKS proof. The main task is to define the traces for the given UML model. For a plain class diagram not containing additional information about the operations, we can only assume that all accessible operations can occur in an arbitrary order. Since this model includes all possible traces, all possibilistic secure information flow properties hold. Therefore we need additional information in the model to do a meaningful analysis. One possibility is to define a state chart, which defines allowed traces. Thus, we use such a state chart in the example (see Fig. 3).

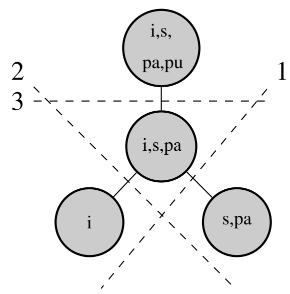
Calculation of the Trace Set: The analysis of SIF properties using MAKS is based on a system model given as set of traces. This set enumerates all possible valid runs of the system. Since UML does not define a standard semantics, we define a semantics for calculating the traces. There are some proposed semantics like dynamic meta modeling [12] or CSP-OZ [13]. UMLsec itself uses a semantics given by a kind of abstract state machines called *UML-machines* [3]. These machines can be naturally interpreted as *stream processing functions* (SPF) which maps possibly infinite streams to other streams. Because derivation of SPFs out of UMLsec is already described in [3], we briefly introduce these functions and calculate the trace set using them.

The semantics of UMLsec is given by a stream processing function $[\![A]\!]_{i,o}(I)$ where A is a class, component or subsystem. The sets i and o denote the input and the output channels. Channels correspond to the operations of a class (or instance of the component or subsystem) and their possible results. The function itself takes an input I modeled as a sequence of input events corresponding to the input channels. The output of the function is a sequence of output events corresponding to the output events. A trace is input/output-alternating ($alt_{i,o}$), if each input event is followed by an output event and vice versa. Each of the events can be an empty event, e.g., Skip. The intention of this property is to preserve the logical order in the trace set. For example every output event must occur after the corresponding input event. Using this we obtain the traces by:

$$Tr := cl_{prefix}\{tr \in E^* | [\![A]\!]_{i,o}(tr|_i) = tr|_o \wedge alt_{i,o}(tr)\}$$

The function cl_{prefix} calculates the prefix closure of the given set.

A trace is in the trace set, if the sequence of the contained input events can produce the sequence of the output event.



i	invoice
s	setCCard
pa	pay
pu	punish

Cut	high	low	
	C	V	N
1	pu,i	s,pa	$\emptyset$
2	pu,s,pa	i	$\emptyset$
3	pu	i,s,pa	$\emptyset$

Figure 9. Calculation of Views

An output event can only occur after the corresponding input event and a stream function always has (a possible empty output) for every input. We model this using the $alt_{i,o}$ operator. This approach ensures that the trace is an alternating sequence of input and output events.

In every trace of Fig. 3 after the first invoice event another invoice event can only appear after at least on pay event has occurred. Between these events there can be arbitrary occurrences of setCCard. Some example traces are:

$$invoice \rightarrow pay \rightarrow invoice \rightarrow pay \rightarrow \ldots$$
$$invoice \rightarrow pay \rightarrow setCCard \rightarrow invoice \rightarrow pay \rightarrow \ldots$$
$$setCCard \rightarrow invoice \rightarrow pay$$

Calculation of MAKS views: Having calculated the traces, we still need the views for the MAKS analysis. The views will be calculated from the security level poset. Fig. 9 shows the principle. On the left side there is a Hasse diagram of the poset defined by the $\langle\langle DefSecurityLevels\rangle\rangle$ stereotype. For readability the security levels are abbreviated. A cut through this poset divides the Hasse diagram into two parts. The events in the lower part are used as visible events. The other events in the graph are treated as confidential. Because we have no unlabeled features in classes, the set N is empty. In the example, there are three valid cuts depicted in Fig. 9, also showing a table with three views for this poset. The reduction of the security levels to basic security levels representation the MAKS views are modeled in the SL profile's analysis model. The cuts are done according to [14].

Now, we combine traces and BSPs to pick up the example: The view is calculated to:

$$\begin{aligned} V &= \{invoice\} \ \ (=\{i\}) \\ N &= \emptyset \\ C &= \{punish, pay, setCCard\} \ \ (=\{pu, pa, s\}) \end{aligned}$$

The BSP named strict removal should be fulfilled. Therefore we check

$$\begin{aligned} SR_{V,N,C}(Tr) &= \forall \tau \in Tr : \tau|_{V\cup N} \in Tr \\ &= \forall \tau \in Tr : \tau|_{\{invoice\}} \in Tr \end{aligned}$$

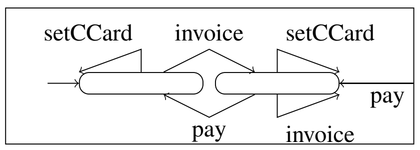


Figure 10. Corrected State Chart

using the trace set calculated before. This BSP is not fulfilled, because no sequence has two consecutive invoice events. This is an unwanted information flow, because a seller gains the information, that a customer has paid all his invoices if the customer can execute the next invoice event. Since this check failed using the first cut of the three cuts, we omit the others. To repair this problem, we change the state chart (Fig. 10).

This ends the example and the description of our approach. We found an issue and repaired it. Using the presented technique we can prove that the corrected model is not leaking unintended information.

In this application we considered a specific security requirement (MAKS style formalization of SIF). What should be done if the requirement evolves to a new version? If the definition of the needed information (here SL and BSP name) does not change, we can simple build a new profile by coping the (unmodified) stereotype profile and build a new analysis model together with a new transformation. Exchanging the profiles takes the changed analysis model in action. In this case the model need not to be adapted to execute the check. If more information is needed, also the syntax need to be adapted. Thus, the separation of syntax and analysis model gives the flexibility for evolving an analysis model as well as a full UMLsec profile.

VI. Related work

There are different approaches to deal with evolution that are related to our work. Within *Software Evolution Approaches*, [15], [16] derives several *laws of software evolution* such as "Continuing Change" and "Declining Quality". [17] argue that it is necessary to treat and support evolution throughout all development phases. They extend the UML meta-model by *evolution contracts* to automatically detect conflicts that may arise when evolving the same UML model in parallel. This work is not security-specific and does not consider analysis models.

Domain specific languages [18], [19] are focusing on a similar problem. They can be used to create a family of languages which are generated at need. Thus they are highly configurable, but due to a missing analysis model they fail to provide a modular and customizable semantics and analysis. Furthermore the use of most existing modeling tools is not possible. The use of a static and not modular meta-model was used in [20], thus the evolution can not be archived easily. Aspect oriented modeling [21], [22] gives a possibility to define internal and external semantics, but focus on code generation and the analysis is not straightforward.

VII. Conclusion

The modular approach and the separation of syntax and analysis model enables us to exchange UMLsec 2.0 profiles. A new profile can be evolved out of an existing by enhancing the syntax and evolution of the analysis model, so that the impact of a modified certification or security requirement is separated from the model. The model can be checked and adapted using the evolved profile. In addition the technique to combine heavyweight and lightweight extensions enables us to use the semantic power of the specialized meta-model but also allows to use modeling tools for the development. Using the separation of the syntax definition in the lightweight extension and the analysis model together with the modularization support the evolution of certifications like security certifications. The main idea is to introduce an analysis model for the representation of the semantic model. Our approach is completed by a transformation from the syntax to the analysis model. We showed the principle using the example of the modeling secure information flow in UMLsec.

Acknowledgement: This work is partially funded by the DFG project "SecVolution" which is part of DFG Priority Program 1593 "Design for Future".

References

[1] M. Lehman, "Software's future: Managing evolution," *IEEE Software*, vol. 15, no. 1, pp. 40–44, 1998.

[2] H. Lipson, "Evolutionary systems design: Recognizing changes in security and survivability risks," Carnegie Mellon Software Engineering Institute, Tech. Rep. CMU/SEI-2006-TN-027, September 2006.

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

[4] H. Mantel, "A uniform framework for the formal specification and verification of secure information flow," Ph.D. dissertation, Saarland University, Saarbrücken, Germany, 2003.

[5] J. Jürjens, "Sound methods and effective tools for model-based security engineering with UML," in *ICSE*, 2005.

[6] "UMLsecTool," 2001–2011, http://jan.jurjens.de/umlsectool.

[7] J. Jürjens, J. Schreck, and P. Bartmann, "Model-based security analysis for mobile communications," in *ICSE*. ACM, 2008.

[8] "CARiSMA," http://vm4a003.itmc.tu-dortmund.de/carisma/web/doku.php (Accessed July 2011).

[9] H. Mantel, "Possibilistic definitions of security – an assembly kit," in *Proceedings of the IEEE Computer Security Foundations Workshop*. Cambridge, UK: IEEE Computer Society, July 3–5 2000, pp. 185–199.

[10] D. E. Bell, "Looking Back at the Bell-La Padula Model," in *Proceedings of the 21st Annual Computer Security Applications Conference*, ser. ACSAC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 337–351.

[11] A. Schürr and F. Klar, "15 years of triple graph grammars," in *Graph Transformations*, ser. LNCS, H. Ehrig, R. Heckel, G. Rozenberg, and G. Taentzer, Eds. Springer Berlin / Heidelberg, 2008, vol. 5214, pp. 411–425.

[12] G. Engels, C. Soltenborn, and H. Wehrheim, "Analysis of UML activities using dynamic meta modeling," in *Formal Methods for Open Object-Based Distributed Systems, 9th IFIP WG 6.1 International Conference, FMOODS 2007*, E. B. Bonsangue, Marcello M.; Johnsen, Ed., vol. 4468. Springer, June 2007, pp. 76–90.

[13] M. Möller, E. Olderog, H. Rasch, and H. Wehrheim, "Integrating a formal method into a software engineering process with UML and Java," *Formal Aspects of Computing*, vol. 20, no. 2, pp. 161–204, March 2008.

[14] H. Mantel, "Talk, RS3 Tutorial," summer 2010 in Buchenau.

[15] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski, "Metrics and Laws of Software Evolution – The Nineties View," in *METRICS'97*. Washington, DC, USA: IEEE Computer Society, 1997, pp. 20–32.

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

[17] T. Mens and T. D'Hondt, "Automating support for software evolution in UML," *Automated Software Engineering Journal*, vol. 7, no. 1, pp. 39–59, February 2000.

[18] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM Comput. Surv.*, vol. 37, no. 4, pp. 316–344, Dec. 2005.

[19] S. Anonsen, "Experiences in modeling for a domain specific language," in *UML Modeling Languages and Applications*, ser. LNCS, N. Jardim Nunes, B. Selic, A. Rodrigues da Silva, and A. Toval Alvarez, Eds. Springer Berlin / Heidelberg, 2005, vol. 3297, pp. 187–197.

[20] J. Eichler, "Lightweight modeling and analysis of security concepts," in *Engineering Secure Software and Systems*, ser. LNCS, U. Erlingsson, R. Wieringa, and N. Zannone, Eds. Springer Berlin / Heidelberg, 2011, vol. 6542, pp. 128–141.

[21] J. Kienzle, J. Gray, D. Stein, W. Cazzola, O. Aldawud, and T. Elrad, "11th International Workshop on Aspect-Oriented Modeling," in *Models in Software Engineering*, ser. LNCS, H. Giese, Ed. Springer Berlin / Heidelberg, 2008, vol. 5002, pp. 1–6.

[22] L. Fuentes and P. Sánchez, "Dynamic weaving of aspect-oriented executable UML models," in *Transactions on Aspect-Oriented Software Development VI*, ser. LNCS, S. Katz, H. Ossher, R. France, and J.-M. Jézéquel, Eds. Springer Berlin / Heidelberg, 2009, vol. 5560, pp. 1–38.