Aggregation of security attributes based on the granularity level of the system

Thesis Submitted in Partial Fulfilment of the Requirements for the Degree

of

Master of Science

to the Department of Computer Science of Freie Universität Berlin

by

Artemij Voskobojnikov
Student ID: 4557770
voskobojnikov.artemij@gmail.com

First Reviewer: Prof. Dr. Jörn Eichler
First Reviewer: Prof. Dr. Marian Margraf

Berlin, <Datum>

Affirmation of independent work

I hereby declare that I wrote this thesis myself without sources other than those indicated herein. All parts taken from published and unpublished scripts are indicated as such.

Berlin, <date>

(Artemij Voskobojnikov)

Acronyms

List of Figures

Abstract

Contents

1 Introduction

Security concepts can be used to capture the interacting system components, potential threats and countermeasures.
For large information systems such concepts can become very large because of the number of the involved sub-systems/components. Interconnectivity and interdepence amongst components may increase the overall system complexity and it might be therefore difficult to detect all potential impacts [2]. Methods for system abstraction that address this problem already exist. The abstraction here is the creation of representation layers which only reflect relevant properties of a system and therefore provide a better level of understanding for the respective user [12]. This for example can be achieved by different projections which reflect different granularity levels of a system displaying different levels of details [16].
In the security context such projections could be used to focus on the security or insecurity of certain sub-systems. Security attributes of components could thus be viewed separately and the security risk for a respective component could be derived. This might become especially useful when the security concept is incomplete or only partially available. An aggregation of security attributes based on the chosen projection will become possible. The system structure could then be used to derive security attributes for components that previously had none. Thus, potentially new information might become processable.
Aggregation methods for security attributes have already been suggested by researchers, e.g. transformation rules for security requirements by Menzel et al. [8] or aggregation rules for attack graphs by Noel et al. [9]. None of those methods take granularity levels or general system hierarchy into account whereas the goal of this thesis is to provide an approach which makes it possible for a user to select a sub-system of interest, i.e. a projection which reflects a certain granularity level and provides the corresponding security attributes. The relevant attributes as well as dependencies and possible aggregations will be shown to ensure an overall complete picture of the selected sub-system. This information can then be used to assess and improve the security level of the selected projection or its dependencies.

2 Background

Prior to addressing the actual approach and implementation some concepts and terms have to be introduced. Firstly, the term *security concept*, as it is used throughout the thesis, is being described. A definition of *granularity levels* and system abstraction follows. Lastly, a section covers *model transformations* and *aggregation rules* on security attributes.

2.1 Security

Morrie Gasser [3] published a book in 1988 providing solutions for computer scpecialists interested in computer security. The term *Computer Security* usually dealt with three aspects:

- Prevention of theft or damage to the hardware
- Prevention of theft or damage to the information
- Prevention of service disruption

With the invention of the Internet the (personal) computers got interconnected and the sharing of data became prevalent and thus data security.
Nowadays security is seen as a process. No combination of products are guaranteed to make a system secure since they are only as secure as the people configuring them [17].
In general when talking about computer security three aspects are addressed, *Confidentiality*, *Integrity* and *Availability* [11]. Each of them will no be defined:

Confidentiality Assets can only be accessed by authorized parties

Integrity Assets can only be modified by authorized parties

Availability Assets are available to authorized parties when needed

The challenge is seen in finding the right balance between the three *Security Goals*. This situation is aggravated by the fact that the goals can overlap, be independent or even mutually exclusive. Charles P. Pfleeger [11] for example, addresses this problem by stating that a strong protection of confidentiality might restrict availability in a system.
Information systems consist of hardware, software and data and it is of high importance for the respective stakeholder to secure the object of interest.

When speaking about securing systems or components we look at two different terminologies, *Vulnerabilities* and *Threats*. A *vulnerability* is a system weakness, be it in implementation or design, that can be potentially exploited by an adversary. Without the intent of exploiting it the vulnerability has no further effect on the system.
A *threat* however is a set of circumstances that has the theoretical potential to cause harm by exploiting a specific vulnerability [17]. An example to demonstrate the differences follows:

Def.1 **Vulnerability**: Sending sensitive data over an unsecured channel

Def.2 **Threat**: Exploiting the unsecured channel by eavesdropping and gathering the sensitive data

As mentioned before a system weakness does not necessarily mean loss of data or any other system breach. Only when coupled with a set of circumstances and the intent of exploitation it turns into a threat. Figure 1 illustrates the four acts that cause security harm as mentioned by Pfleeger [11].

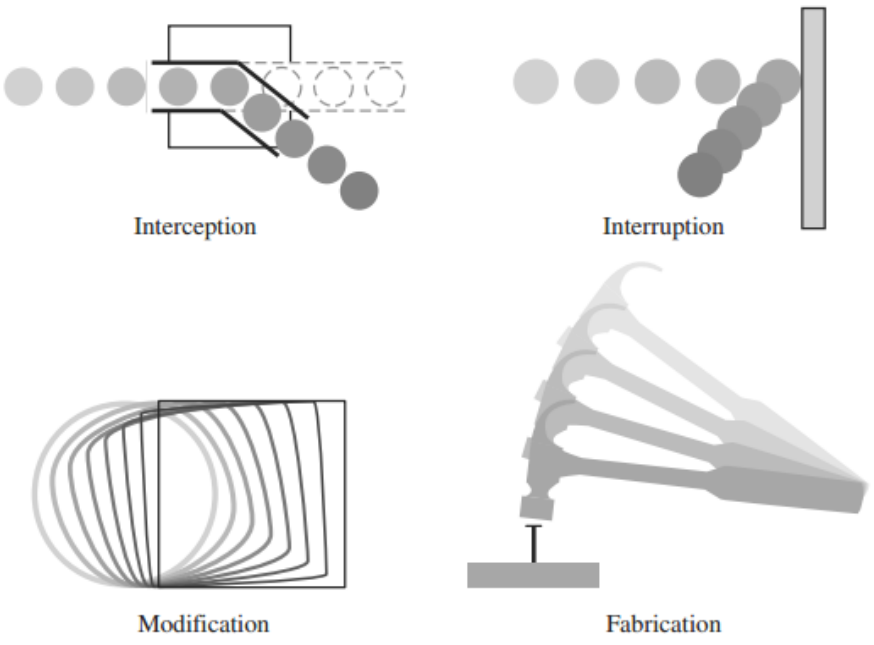

Figure 1: Four acts that cause security harm

Confidentiality, integrity and availability, or the C-I-A triad like it is also being called, can be viewed from a perspective of the causes of harm. Confidentiality can suffer if some unauthorized party intercepts the data, availability

can be lost if a flow of data is being interrupted and lastly integrity can be broken if the data is being modified or false data is being fabricated by an adversary.
To prevent the vulnerabilities from being exploited and the threats from potentially causing harm one can use *Controls* or *Countermeasures* to secure parts of a system. An example would be the use of encryption to prevent sensitive data from being eavesdropped on as mentioned in **Def.1**.
To put everything into perspective the term *Security Architecture* will be introduced.

2.1.1 Security Architecture

Gacek et al. discussed the definition of a *Software System Architecture* (SSA) which will be used and adapted in the security context. According to the authors a *Software System Architecture* is:

- A collection of software and system components, connections and constraints
- A collection of system stakeholders' need statements
- A rationale which demonstrates that the elements which define a system satisfy the the stakeholders' needs, if implemented correctly

Here, a *Security Architecture* (SA) would be an adapted definition where, if implemented correctly, the stakerholders' *security* needs will be satisfied. Naturally, the connections as well as the respective components might differ from the Software System Architecture as they might have to be enhanced or altered to satisfy the security needs.
The stakeholders themselves however, are very similar. Gacek et al. differentiate between five parties, the *Customer*, the *User*, the *Architect and System Engineer*, the *Developer* and the *Maintainer*.
The International Standard ISO 27001 provides a model for establishing, impelementing, operating, monitoring, reviewing, maintaining and improving an organization's Information Security Management System [1], i.e. a system that is responsible for establishing and maintaining organization's security needs and guidelines. When comparing these rather short definitions one can already see an overlap when looking at potential stakeholders. An overview on the SSA stakeholders follows:

Stakeholder	SSA Concern
Customer	Schedule and budget estimation Feasibility and risk assessment Requirements traceability Progress tracking
User	Consistency with requirements and usage scenarios Future requirement growth accommodation Performance, reliability, interoperability
Architect and System Engineer	Requirements traceability Support of tradeoff analyses Completeness, consistency of architecture
Developer	Sufficient detail for design Reference for selecting/assembling components Maintain interoperability with existing systems
Maintainer	Guidance on software modification Guidance on architecture evolution Maintain interoperability with existing systems

When looking into the International Standard, one can see that similar concerns can be also found in the adaptation of the „Plan-Do-Check-Act“ model for the establishment of an ISMS process. In the case of ISMS similar aspects can be found in the four different phases as seen in Figure 2.

Plan (establish the ISMS)	Establish ISMS policy, objectives, processes and procedures relevant to managing risk and improving information security to deliver results in accordance with an organization's overall policies and objectives.
Do (implement and operate the ISMS)	Implement and operate the ISMS policy, controls, processes and procedures.
Check (monitor and review the ISMS)	Assess and, where applicable, measure process performance against ISMS policy, objectives and practical experience and report the results to management for review.
Act (maintain and improve the ISMS)	Take corrective and preventive actions, based on the results of the internal ISMS audit and management review or other relevant information, to achieve continual improvement of the ISMS.

Figure 2: ISMS Stakeholder Concerns

Security requirements and objectives have to be defined in the earlier phases so they can be planned, implemented and carried out by the respective security architects/developers/security engineers. Here, the term user is not as clear since a variety of employees can be seen as such, e.g. any employee that accepts the established security policies. Since systems, as mentioned

in Section 2.1, are just as secure as the people using them, many employees can be viewed as ISMS users. They would be interested in implementing and operating the processes and procedures which have been defined in the previous phase.
Established security policies have to be also monitored and maintained. Security is never guaranteed and is a complex process [17] - systems have to be constantly updated and possibly upgraded.
The needs of the variety of security stakeholders have to be considered throughout the system processes and have to be registered in some way. Security requirements have been mentioned and, amongst others, will be described more thoroughly in the next section.

2.1.2 Common Criteria

The following section will present a way of modeling security concerns for an asset of interest.
Common Criteria proposes an evaluation by using a so called *Security Target* (ST), a construct that encapsulates the *Target of Evaluation* (TOE), threats to the TOE and countermeasures [15]. The goal of the evaluation is to show that the used countermeasures are sufficient to counter potential threats and thus implying that the TOE is sufficiently protected.

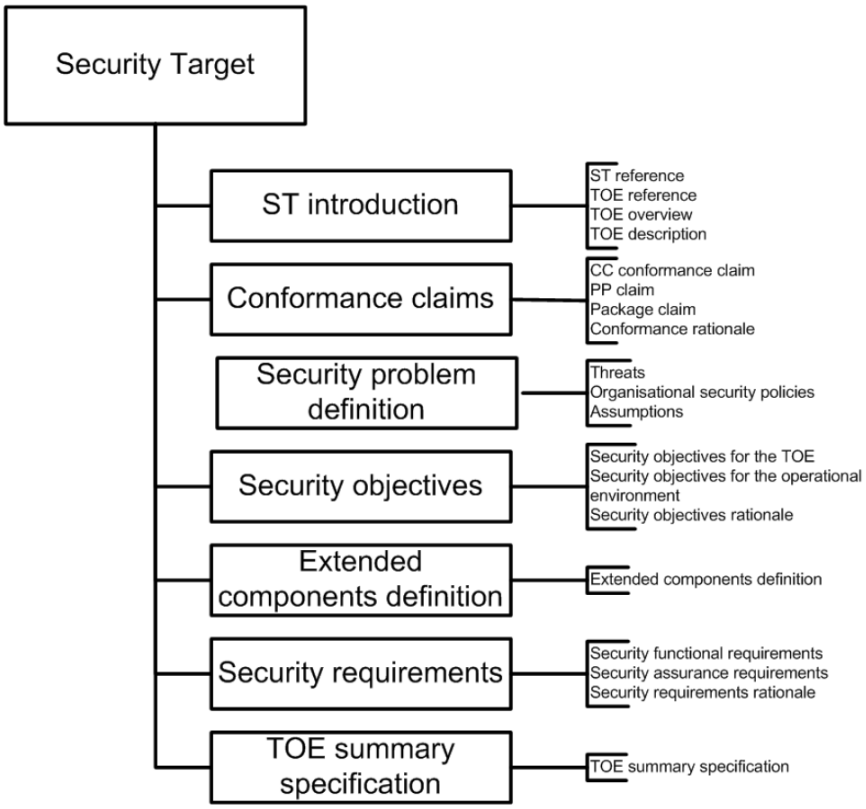

Figure 3: Overview of the Security Target contents

A description of all the contents of a ST is unnecessary here and only the key security attributes of a ST that will be used to construct a *Security Concept* (Subsection 2.1.3) are being introduced.
The *Security Problem Definition* defines, as the name suggests, the security problem that is being addressed. Apart from containing guidelines and assumptions it contains *Threats* which are *„[...] adverse actions performed by a threat agent on an asset“* ([1], p. 66).
A *Security Objective* is an abstract solution to the previously defined security problem. There exists a possibility to divide the *Security Objectives* into part wise solutions, one being the *Security Objectives for the TOE* and the other being the *Security Objectives for the Operational Environment*. Moreover does the ST contain traces showing which objectives address which threats, guidelines and assumptions and a set showing that all threats, guidelines and assumptions are addressed by the security objective.
Security Functional Requirements (SFR) are a more detailed translation of the previously defined *Security Objective*. Despite being more detailed, SFR

have to be still independent from specific technical solutions.
Lastly, STs contain a TOE summary specification where it is stated how the TOE meets all the SFRs and how exactly those requirements are met on a technical level.

2.1.3 Security Concept

The term *Security Concept*, as it is defined here, is based on the constructs introduced in the previous chapters, namely *Security Architecture* and *Security Target*. An overview follows.
Assets are the to be secured objects of interest, i.e. TOE according to Common Criteria. *Assets* can be either logical or physical and can be grouped to sets, if needed.
A *Security Goal* (SG) is the equivalent to the *Security Objective*. A valid SG must address an *Asset* and a *Security Goal Class* that defines the actual purpose of the SG. In general the set of *Security Goal Classes* consists of *Confidentiality*, *Integrity* and *Availability* but can also be expanded by further classes such as *Authenticity*.
Threats serve the same purpose as proposed by Common Criteria. They are adverse actions performed by an entity against an *Asset*.
This information is all brought together in *Security Requirements* that are defined in natural language and show the interrelationships between elements. A *Security Goal* has to be mentioned as well as an *Asset* and a *Threat* against which the object of interest should be protected.
Lastly, *Controls* are the technical measures that counter or minimize the *Threats*.
The Table 1 depicts the relationships between all the security attributes:

2.2 Modeling

To ensure a viable solution one has to think of a representation of real life systems. *Models* can be used to achieve this by depicting the key properties and processes of a certain system. According to Ed Seidewitz [14] a model is a *„set of statements about some system under study“* with the statements being either correct or incorrect.
A system modeled using the Unified Modeling Language (UML) serves as an example. In this case such statements could be made on the relationships between classes and would only be correct if they are consistent with the actual structure of the respective system under study (SUS), i.e. the described (modeled) relationships do indeed exist.

Name	Contains	Description	Example
Asset	-	Digital or physical object of interest that should be secured	Sensible user data
Security Goal Class	-	Defines the purpose of the Security Goal	Confidentiality of sensible user data
Security Goal	Security Goal Class, Asset	Defines the security objective	Confidentiality of sensible user data shall be protected
Threat	Asset	Adverse action against an Asset	Eavesdropping on sensible user data
Security Requirement	Asset, Security Goal, Threat	Security Objective in natural language	The Confidentiality of sensible user data shall be protected against eavesdropping
Control	Threat	Measure to minimize or mitigate the Threat	Encryption of sensible user data with AES-256 to prevent eavesdropping

Table 1: Elements of a Security Concept

In our case we would try to create a model that reflects the security attributes and their interrelationships in a SUS. This interpretation of a model is key because only then the model is given a meaning [14].
A definition of a model is not enough. A *metamodel* has to be clearly defined to verify whether a model is conform or not, i.e. whether a security concept instance is conform to its security concept metamodel. The following figure shows the interrelationships.

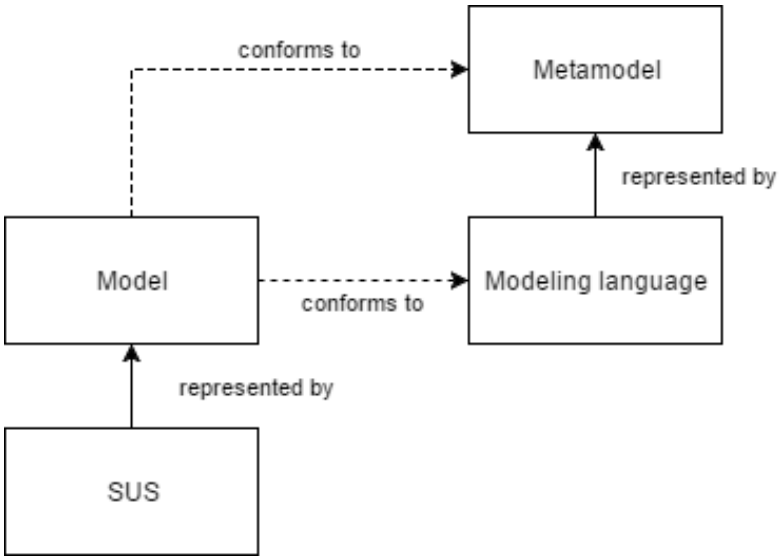

Figure 4: Relationships between Model and Metamodel

A security concept of a SUS would be modeled in a modeling language, e.g. UML which is a representation of its own metamodel. At the same time the security concept would be conform to its metamodel. This conformity, be it the security concept or the modeling language, is needed for a model to be considered valid.

2.2.1 Model Transformation

The Meta Object Facility (MOF) [10] is a standard metametamodel proposed by the Object Management Group (OMG) and captures the relationships between models in a three-layered architecture consisting of M1, M2 and M3. Models (M1) are representations of systems and are expressed in a modeling language M2, e.g. UML as mentioned in the previous section, which is conform to a so called metamodel. Metamodels themselves are also expressed in a metamodeling language which is conform to a metametamodel (M3).

These architecture levels can be found in the *model transformation pattern* by Jouault et al. [4] which can be seen in Figure 5.

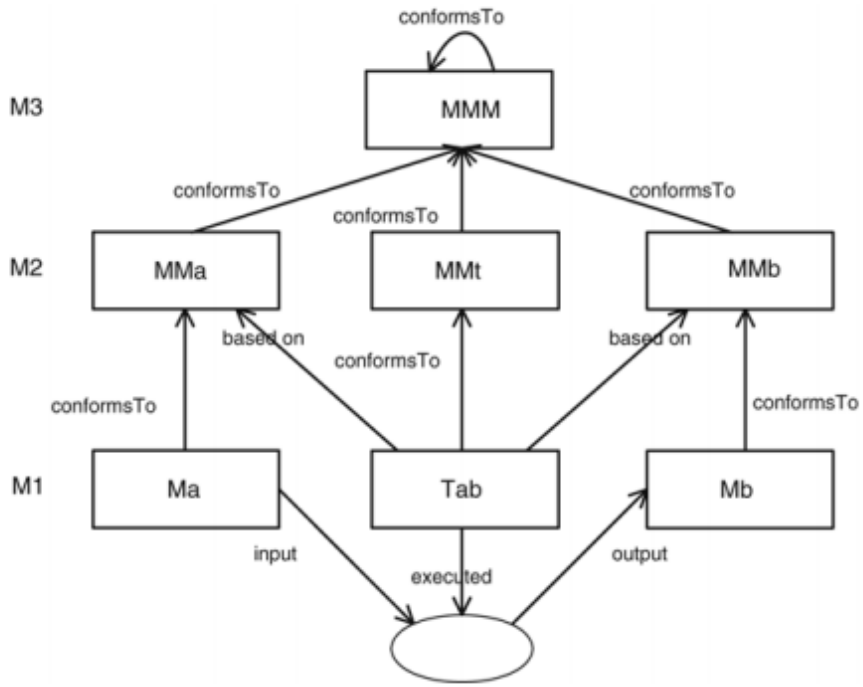

Figure 5: Model transformation according to Jouault et al.

Here a source model Ma is being transformed into a target model Mb using a transformation language. Both models and the transformation language are conform to their respective metamodel which is the traditional understanding of a model transformation.
Kleppe et al. defined a *transformation* as an automatic generation of a target model from a source model according to *transformation rules* that describe how elements from the source model can be transformed into a target model [6].
This transformation may have different levels of automation. An *automatic* transformation would not need any manual intervention from the user. In cases where incompleteness or inconsistencies may occur a manual intervention may be needed [7].
Throughout the introduction and the background chapter the derivation of security attributes based on structural properties of a system of interest was mentioned. Given a model M this derivation can be seen as alteration of M and therefore as a *model transformation*. The resulting model M' is different

to M, both however, are conform to the same metamodel MM whereas MM is conform to MMM. In our case both source and target languages are identicial. We can therefore simplify the transformation graph shown in Figure 5.

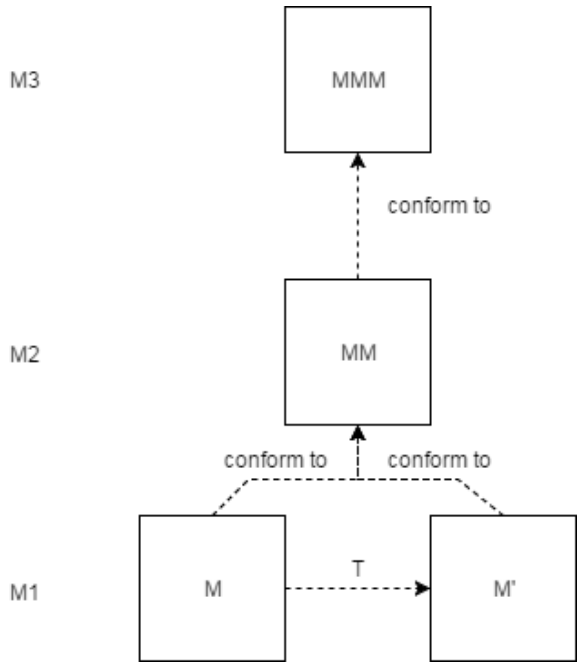

Figure 6: Model transformation

The definition of a transformation T, or better a *transformation rule set*, that alters a model M is the main goal of this thesis.
Prior to the actual rule set definition one final concept has to be introduced. A user-selected *Granularity level* serves as a second input in the model transformation. The transformation itself should be automatic, the only user input should be the just mentioned granularity level.

2.2.2 Granularity Levels

Information systems are often complex because of the number of interconnections and interdependencies between components and therefore it might be difficult to assess potential impacts and risks of a system [2].
One logical goal would be to decrease the overall complexity to enable a better risk assessment. *System abstraction* tries to achieve this by reducing the level of details [2]. Branagan et al. differentiate between two possibilities, *Whole-part decomposition* which decomposes a system into smaller subsystems and

Distinct development perspectives which focuses on certain parts of a system depending on the current development perspective.
In this thesis the focus will be on the *Whole-part decomposition* even though an adaption of development perspectives is certainly possible. The main difficulty would be the definition of such perspectives in the security context because they would be highly dependent on the respective security analyst. The perspectives would not be unambiguous.
Therefore the change of the level of detail, i.e. the granularity level, will be achieved by uniting or decomposing components of a system of interest. By decomposing a larger system into smaller subsystems one could focus on only specific security attributes and dismiss others.
A user will select a certain granularity level, i.e. a certain set of components, as an input to the transformation function T as mentioned in 2.2.1. Together with the defined rule set a valid model M' will be generated which to be considered valid has to be conform to its metamodel MM.

3 Related Work

Several publications that provide an essential basis for this thesis will now be presented. A variety of different topics will be briefly covered such as abstraction layers/granularity layers, security attribute aggregation and propagation as well as model transformations and formal verification.

4 Approach

This section presents the approach addressing the previously mentioned goal of a model transformation based on the user-selected granularity level of a system of interest. Firstly, the security concept metamodel will be thoroughly described, each element of the metamodel will be put in the security context and advantages and disadvantages of such an interpretation will be highlighted.
The second part will deal with the actual transformation rules. Model transformations will be mathematically defined and the transformation rules for each element of the model will follow. Aside from the solution possible edge cases will be presented and evaluated.

4.1 Security Concept

The following metamodel is based on the security elements mentioned in Section 2.1.3. It shows the interconnections between elements and adds re-

strictions. This metamodel serves as a base enabling the creation of model instances capturing the relations between components/assets of a specific SUS. It also provides a security context and the option for the user to select a granularity level.

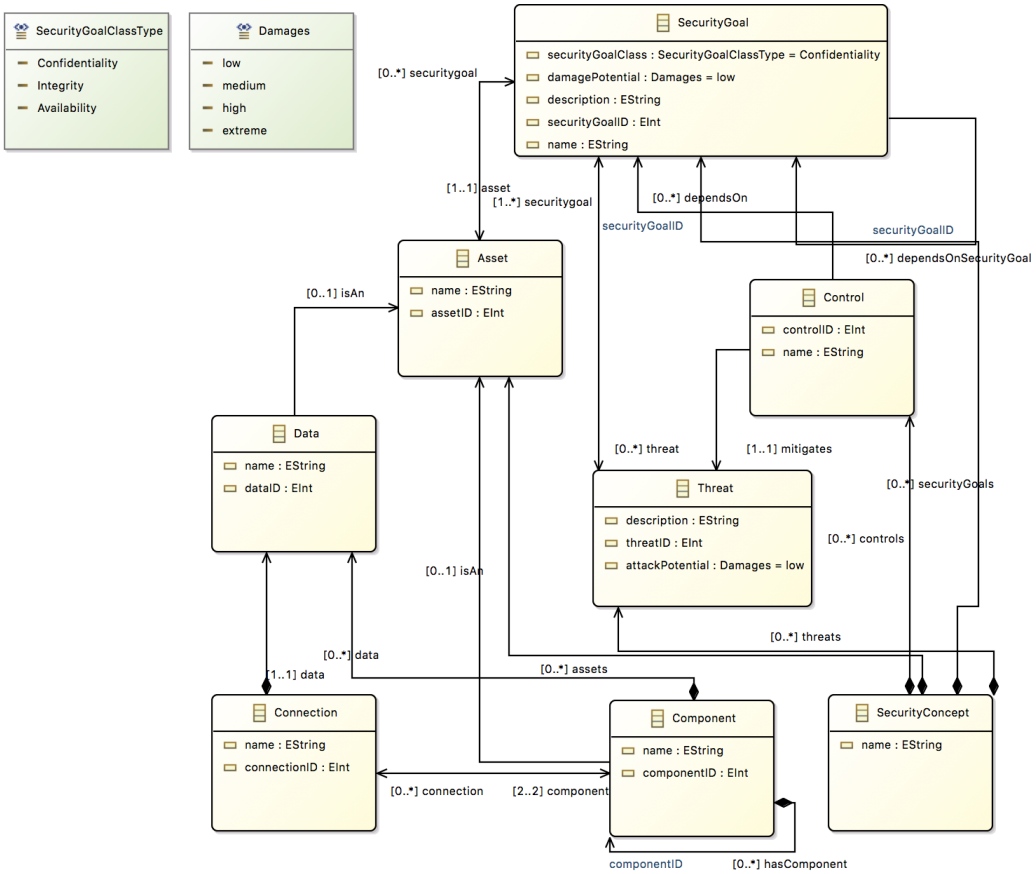

Figure 7: Security Concept Metamodel

In the figure above all the core elements *Security Goal*, *Asset*, *Control* and *Threat* are pictured. All of these elements are part of a *SecurityConcept*, which in this model will be simply identified by a name.
SGs have a *Security Goal Class* attribute which describes the purpose of each SG. The *Damage Potential* attribute indicates at the importance of a Goal, i.e. how important it is to secure a certain asset. The higher the damage potential the higher the impact if the security of an asset is breached. One key aspect of this metamodel is the dependency between SGs. A SG is dependent on another SG if both belong to the same asset and have the same security goal class. These dependencies, amongst others, have to be

considered during potential transformation steps (Section 7).
Each SG belongs to exactly one asset whereas an asset itself can have unlimited SGs. In this thesis both physical and virtual components can be considered an asset. Both *Data* and *Component* can be assets according to the metamodel.
There will be two different types of data. For once *processed data*, i.e. data that is being processed or kept in storage by a specific component. On top of that *transmitted data* will be considered separately since the transmission channel itself can be seen as an asset. The resulting interconnections are shown in the following figure:

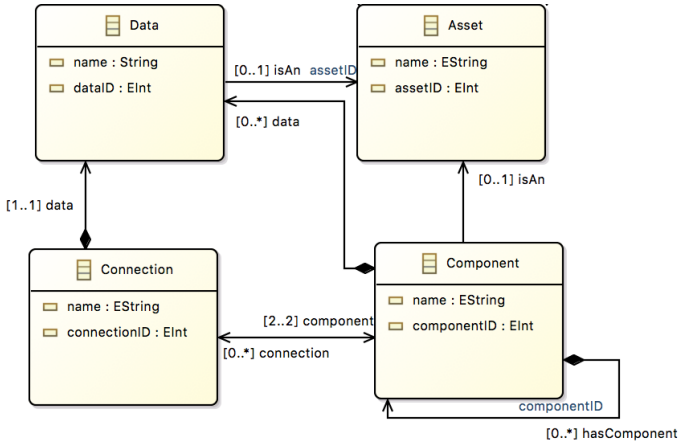

Figure 8: Two different interpretations of data

As already mentioned data can be viewed as being processed and transmitted. Therefore an element *Connection* was added. A connection is the transmission channel between two components. It must have an associated data. The processed or stored data however can be directly associated with a component. In both cases data can be viewed as an asset. There is no possibility to assign a connection as an asset, the reason being that the transmission medium itself, i.e. the cable, wire, is rarely an object of interest but more so the data which is being transmitted.
Lastly the selection of *Granularity Levels* by users should be enabled. Instead of having two different input models, one security concept model and one model depicting the system structure, one can reproduce the structural properties by adding a reference to the component element.

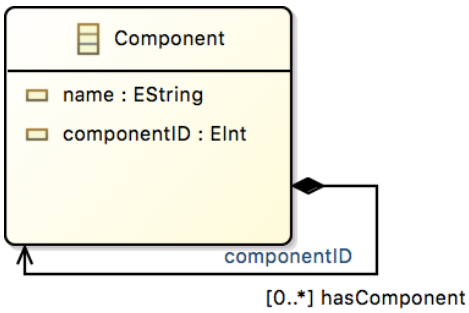

Figure 9: Structural information

Having this extra reference one can create infinitely deep structural dependencies within the model. Therefore the actual transformation will only require one model instead of two separate ones.

4.2 Metamodel elements and their security properties

To put the different elements of the metamodel into a security context one has to clearly define the Security Goal Classes for possible assets. Interpretations of the classes are not unambiguous and it is necessary to discuss how security attributes propagate through different abstraction layers and what kind of impact interrelated components on different layers have on each other.

4.2.1 Components

Before introducing the transformation rules one has to look at the different kinds of components that can be potentially found in a model instance. Even though the model element remains the same (*Component*) a distinction which is made here is necessary because the interpretation of Security Goal Classes differs depending on the component type.

Physical Component

Physical components are components that can be accessed physically, e.g. computers, servers, switches etc. Since those can be accessed physically and therefore be manipulated physically one has to define the Security Goal Classes accordingly.

1. **Confidentiality** - Will be *undefined* for physical components and will only hold for data stored/processed on those components

2. **Integrity** - Ensured when the component has not been altered by an adversary in any way; can be broken by an adversary having physical access

3. **Availability** - Ensured when the component is able to carry out its designated task; can be broken by an adversary having physical access

Virtual Component

Virtual components cannot be directly accessed physically. Examples would be virtual machines, virtual switches etc. The classes are similar to the physical counterpart except from the breach of the respective class.

1. **Confidentiality** - Will be *undefined* for virtual components and will only hold for data stored/processed on those components

2. **Integrity** - Ensured when the component has not been altered by an adversary in any way; can be broken remotely by an adversary, i.e. without having physical access

3. **Availability** - Ensured when the component is able to carry out its designated task; can be broken remotely by an adversary, i.e. without having physical access

4.2.2 Data

Similar to the component element the data element can be interpreted in different ways. The distinctions between the different data types are very minor but noteworthy nonetheless. We consider three types of data, data which is being stored by a component, data which is being processed by a component and data which is being transmitted between components. The common terminology is *Data at Rest*, *Data in Motion* and *Data in Use* [5]. The used terms might be new, the distinction between different states of data however can be found in publications going as far back as 1982, e.g. by Dorothy E. Denning [13].

Data at Rest

Data at Rest applies to devices that hold data [5]. An example would be a database containing sensitive information. The database being the *Virtual Component* and the stored data being the *Data at Rest*. One could also define the server with the database as a *Physical Component*.

1. **Confidentiality** - Ensured when the data is protected from unauthorized disclosure at rest
2. **Integrity** - Ensured when the data is protected from unauthorized modification at rest
3. **Availability** - Ensured when the data is available to authorized parties when needed, i.e. authorized parties can access and use the data when needed

Data in Use

Data in Use applies to data that is being processed by a component, i.e. is being used by a service running on a component. An example would be data that is being processed by an API on a server. The API being the *Virtual Component* and the server being the *Physical Component*. The *Data in Use* would be the processed information by the API backend.

1. **Confidentiality** - Ensured when the data is protected from unauthorized disclosure during processing
2. **Integrity** - Ensured when the data is protected from unauthorized modification during processing
3. **Availability** - Ensured when the data is available to authorized parties when needed, i.e. the data is being processed by the service/the respective service is running when needed

Data in Motion

Data in Motion applies to all data transmitted between components. We do not specify any protocols here to keep the definition as broad as possible.

1. **Confidentiality** - Ensured when the data is protected from unauthorized disclosure during transmission
2. **Integrity** - Ensured when the data is protected from unauthorized modification during transmission
3. **Availability** - Ensured when the data is available to authorized parties when needed, i.e. is not lost or intercepted during transmission

The different component/data types shown here should only show the different interpretations of the elements of the security concept metamodel. The interpretation of the element itself does not have an influence on the chosen metamodel element, i.e. the element for both physical and virtual components will still be *Component*. The only difference can be found in data since transmitted data always belongs to a *Connection*. For both processed and stored data however the metamodel element *Data* will be used.

4.2.3 Interpretation of Interconnections between Elements

To describe model transformations in the security context one has to discuss how exactly the security attributes propagate through the respective abstraction layers. Here we will look into the dependencies between components and their security properties during transformations.

Sub-component

A sub-component SC will be defined as a component that directly belongs to a component C which is in an abstraction layer above, i.e. encapsulates one or more sub-components. According to the previously defined metamodel such a relationship between components is being defined by the *hasComponent* composition, i.e. a sub-component cannot exist without a component here. This also means that no new data types or structures are added when changing to higher granularity levels. Sub-components can only process data that has already been defined in the layers above. An example would be a database encapsulating a sub-component such as a secure key storage.

Security Goals

For security goals the interconnections between components and sub-components have to be interpreted in an unambiguous way. There is no clear definition on how SGs for components in higher abstraction layers propagate to the respective sub-components in the abstraction layers below. We therefore adopt the idea for security requirements which was proposed by Menzel et al. [8]. Here, the main focus will be on two characteristics, *Independent* and *Require* to capture interdependencies between SGs on different granularity levels. According to the metamodel each SG has the following attributes:

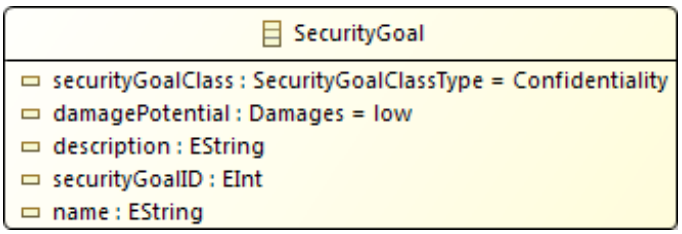

Figure 10: Attributes of a Security Goal

The key attributes are *securityGoalClass* and *damagePotential*. Both are relevant when it comes to aggregation rules and dependencies amongst components. In this chapter the focus will be on the latter.
In case of a complete security concept definition interdependencies amongst components and security goals are clear, similar to Figure 11. Out of simplicity security goals will be linked directly to components and not through assets.

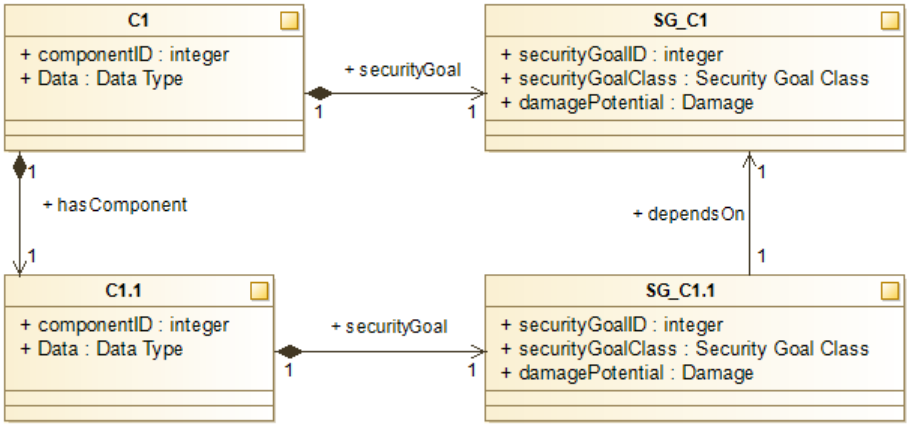

Figure 11: Relationships between Security Goals

Figure 11 has shown the dependencies between components and their security attributes in case of a complete security concept definiton. In underspecified security concepts, such as shown in Figure 13, the dependencies are not as trivial.

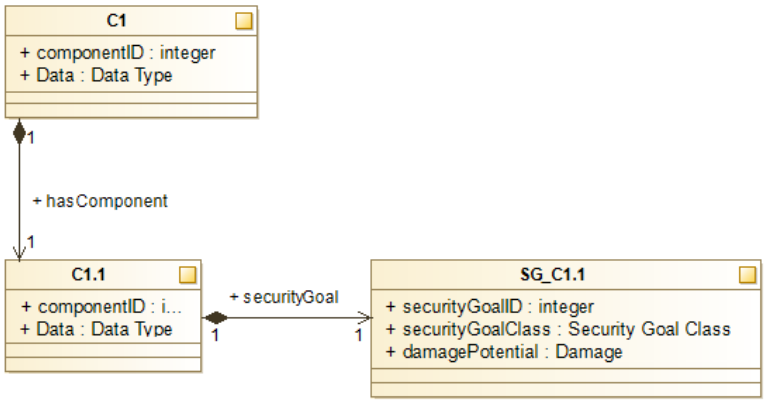

Figure 12: Ambiguous propagation

It is not obvious what kind of influence the sub-component has on its parental node, if at all. If the confidentiality of component $C_{1.1}$ is protected it is not clear how it will propagate to higher abstraction layers. Similar observations can be made for the aggregation from higher abstraction layers to lower ones. One has therefore to define when and how security attributes will propagate based on structural features of the SUS.

Definition 1 *A security goal SG_{C_1} of a component C_1 has a direct influence on another component C_2 if and only if there is:*

1. *a direct link between the two components, i.e. a hierarchical relationship between the two components AND*

2. *component C_2 processes/holds the data type that is being addressed by SG_{C_1}*

This *direct influence* is one of the more obvious dependencies amongst components. At first glance matching data types are needed to aggregate SGs through different abstraction layers but it is certainly possible that security concepts might be underspecified. Relying on this definition might therefore severely limit the transformation and thus, the resulting model.
An examplary model instance follows showing security attribute propagation with an underspecified security concept. Similarly to previous examples data/components will be directly linked with security goals without asset elements in between in the interest of greater clarity.

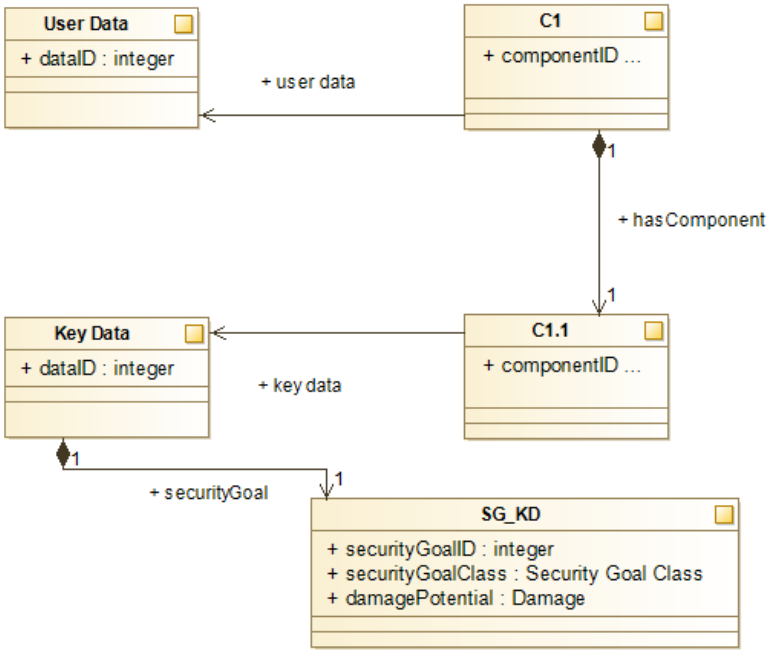

Figure 13: Ambiguous propagation

The corresponding security goal of the key data in natural text could be:

SG_{KD} : The Confidentiality of sensible key data should be protected

Now we will look at component $C_{1.1}$ which has a link to key data. When looking through its SGs we will only find SG_{KD} which should protect the *Confidentiality* of said data. There is also a direct connection from $C_{1.1}$ to component C_1 which is in a granularity level above.
According to the definition (Section 4.2.3) a sub-component can only process data that already exists in the abstraction layers above. Thus, a correct conclusion would be that if the confidentiality of key data is protected in $C_{1.1}$ it is also protected in the layer above even though there is no explicit link between C_1 and key data. This however, does *NOT* mean that the overall confidentiality is being protected since C_1 may process other data, such as user data in the example.
Propagation from higher layers to lower ones is generally not possible without explicit links to data types. This will be discussed more thoroughly in Section 4.4.
Similar conclusions can be made for the *Integrity* of sub-components and its propagation to abstraction layers with higher or lower granularity levels with one slight difference. When looking at components at higher abstraction

layers we can say that if the integrity of a component C_1 is being protected then all of its sub-components are being protected as well.
For *Availability* however, the situation differs. The connection between components (*hasComponent*) has been defined as a composition. This means that the availability of a specific sub-component is pre-determined by the structural features of the SUS. If the availability of a component C_1 is protected it necessarily means that the availability of its sub-components is protected as well. To have the availability of a component protected however all of the sub-components have to be protected individually. The availability of a component is therefore the union of availabilities of its sub-components.

Threats

Similar to SGs one can look into propagation rules for threats. Firstly we will look at the important attributes of a threat. The key attribute here will be *attackPotential* which will be considered during the transformation steps in Section 4.4.

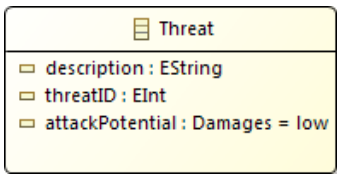

Figure 14: Attributes of a Threat

Here, the focus will be on the propagation of threats through granularity levels based on an underspecified security concept.
Every defined threat threatens at least one SG according to the metamodel (Figure 7) and since every SG addresses one asset (component or data) we can adapt the observations made in the previous section.

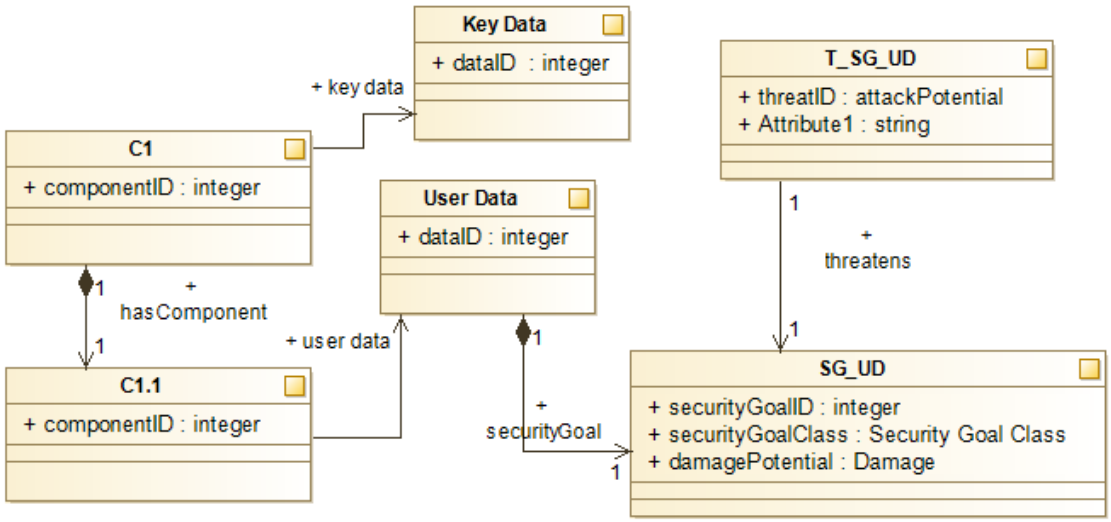

Figure 15: Underspecified Security Concept with a Threat

In Figure 15 a threat $T_{SG_{UD}}$ threatens SG_{UD} which addresses the user data that is being used by sub-component $C_{1.1}$. The influence of T_{KD} on higher abstraction layers is of interest since neither SGs nor threats are explicitly defined in the example.

SG_{UD} : The Confidentiality of sensible user data should be protected

$T_{SG_{UD}}$: Eavesdropping on sensible user data

Similar to SGs, we can assume that $T_{SG_{UD}}$ propagates to the layer above based on the definition of sub-components. User data, which is only explicitly addressed by SG_{UD}, is also present at parental nodes. Thus, if the *Confidentiality* of user data is being threatened by $T_{SG_{UD}}$, it is also being threatened when looking at $C1$.
For the propagation from abstraction layers depicting lower granularities (C_1) to layers with higher granularity ($C_{1.1}$) we have to take the data types into account. We will define a SG addressing key data and the corresponding threat.

SG_{KD} : The Confidentiality of key data should be protected

$T_{SG_{KD}}$: Disclosure of key data

Now if the the confidentiality is being threatened by $T_{SG_{KD}}$ it is not clear how it will propagate to the layers below since $C_{1.1}$ is not addressing key data in any observable way.

Threats that threaten SGs with *Integrity* as their security goal classes behave differently. Contrary to confidentiality, assumptions can be made when looking at propagation from higher abstraction layers to lower ones. When integrity of a component C_n is being threatened the integrity of all respective sub-components $C_{n.1}...C_{n.m}$ is being threatened as well. If the integritiy of a sub-component $C_{n.i}$ is being threatened it will also propagate to its parental node C_n since a node can only be as secure as its sub-nodes.
Lastly, for *Availability* the propagation is pre-determined by the structural features and the definition of sub-components. If the availability of a component C_n is threatened it propagates to its sub-components $C_{n.1}...C_{n.m}$.
If the availability of one sub-component is being threatened it will also propagate to its parental nodes. This however does not provide information on how severely the availability of C_n as a whole will be impaired in case of an attack on the availability of $C_{n.i}$. This propagation from sub-components to their parental nodes is very similar to the just mentioned integrity behavior. The availability of C_n is a union of availabilities of its sub-components and one threat on a specific $C_{n.i}$ would alter said set.
We can therefore conclude that to ensure a SG of a component C_n all its sub-components have to be protected as well. For threats however it is enough to threaten one sub-component to have an effect on its parental node.

Controls

Controls try to mitigate threats and are directly linked to at least one threat at all times. The propagation of controls through abstraction layers will therefore be closely coupled with threats and SGs.

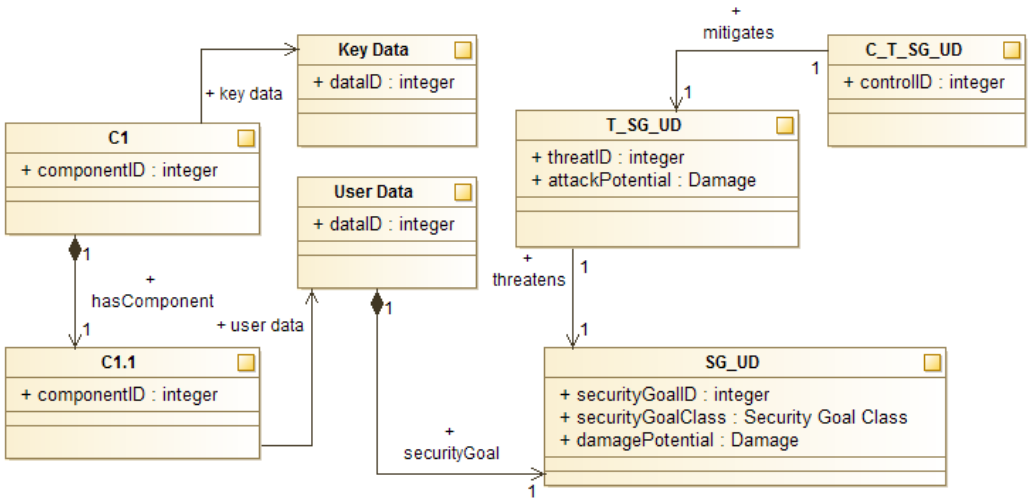

Figure 16: Underspecified Security Concept with a Control

We have already discussed the propagation of SGs and threats in the previous chapters and thus, indirectly the propagation of controls. The example in Figure 16 shows a control that mitigates the threat $T_{SG_{UD}}$. In natural text the security attributes could be the following:

SG_{KD} : The Confidentiality of sensible user data should be protected

$T_{SG_{KD}}$: Eavesdropping on sensible user data

$C_{T_{SG_{KD}}}$: Encryption of sensible user data with AES-256 to pevent eavesdropping

Similar to threats which propagate to the upper abstraction layers, controls will as well. The rules will be the same since the controls are very tightly coupled and cannot exist without threats.
For this example this would mean that when looking at the abstraction layer above and at C_1 we will consider the control $C_{T_{SG_{KD}}}$ as well the threat $T_{SG_{KD}}$ it is mitigating.
This section was only giving an overview on the conclusions for security attributes that can be made based on the structural features of an (underspecified) security concept definition. A more thorough definition will be discussed in the following section.

4.3 Model Transformation

Let SG_{CIA} be the set of security goals for a SUS. A security goal SG is defined as $SG(cl, asset, dmg)$ where cl is the security goal class (C for confidentiality,

I for Integrity and A for availability), *asset* being the element of interest which the SG was defined for and *dmg* is the damage potential in case of a security breach (H = high, M = medium and L = low).

4.4 Transformation Rule Set for Security Goals

4.4.1 Aggregation Rules

4.5 Transformation Rule Set for Threats

4.5.1 Aggregation Rules

4.6 Transformation Rule Set for Controls

4.6.1 Aggregation Rules

5 Implementation

5.1 Technology

5.2 Validity & Verification

5.3 Runtime & Complexity

6 Summary

7 Outlook

References

[1] ISO/IEC 27001:2005 - information technology – security techniques – information security management systems – requirements. Technical report, 2005.

[2] Mark Branagan, Robert Dawson, and Dennis Longley. Security risk analysis for complex systems. pages 1–12, 2006.

[3] Morrie Gasser. *Building a Secure Computer System*. Van Nostrand Reinhold Co., New York, NY, USA, 1988.

[4] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. Atl: A model transformation tool. *Science of Computer Programming*, 72(1):31 – 39, 2008.

[5] Prathaben Kanagasingham. Data loss prevention. *SANS Institute. Aug*, 2008.

[6] Anneke G. Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[7] Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152:125 – 142, 2006. Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005).

[8] Michael Menzel, Christian Wolter, and Christoph Meinel. *Towards the Aggregation of Security Requirements in Cross-Organisational Service Compositions*, pages 297–308. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[9] Steven Noel and Sushil Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, VizSEC/DMSEC '04, pages 109–118, New York, NY, USA, 2004. ACM.

[10] OMG. OMG Meta Object Facility (MOF) Core Specification, Version 2.4.1, June 2013.

[11] Charles P. Pfleeger and Shari Lawrence Pfleeger. *Security in Computing (4th Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2006.

[12] Klaus Pohl, Harald Hönninger, Reinhold Achatz, and Manfred Broy. *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*. Springer Publishing Company, Incorporated, 2012.

[13] Dorothy Elizabeth Robling Denning. *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc., 1982.

[14] E. Seidewitz. What models mean. *IEEE Software*, 20(5):26–32, Sept 2003.

[15] The Common Criteria Recognition Agreement Members. Common criteria for information technology security evaluation. http://www.commoncriteriaportal.org/, September 2012.

[16] Judith Thyssen, Daniel Ratiu, Wolfgang Schwitzer, Alexander Harhurin, Martin Feilkas, and Eike Thaden. A system for seamless abstraction layers for model-based development of embedded software. In *Software Engineering (Workshops)*, pages 137–148, 2010.

[17] J.R. Vacca. *Computer and Information Security Handbook*. Elsevier Science, 2012.