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 All parts taken from published and unpublished scripts are indicated as such.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 [15]. This for example can be achieved by different projections which reflect different granularity levels of a system displaying different levels of details [19].
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. [10] or aggregation rules for attack graphs by Noel et al. [12]. 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 [4] published a book in 1988 providing solutions for computer specialists 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 correspondingly also 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 [21].
In general when talking about computer security three aspects are addressed, *Confidentiality*, *Integrity* and *Availability* [14]. Each of them will now 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 [14] 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 [21]. An example to demonstrate the differences follows:

> **Vulnerability**: Sending sensitive data over an unsecured channel
>
> **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 [14].

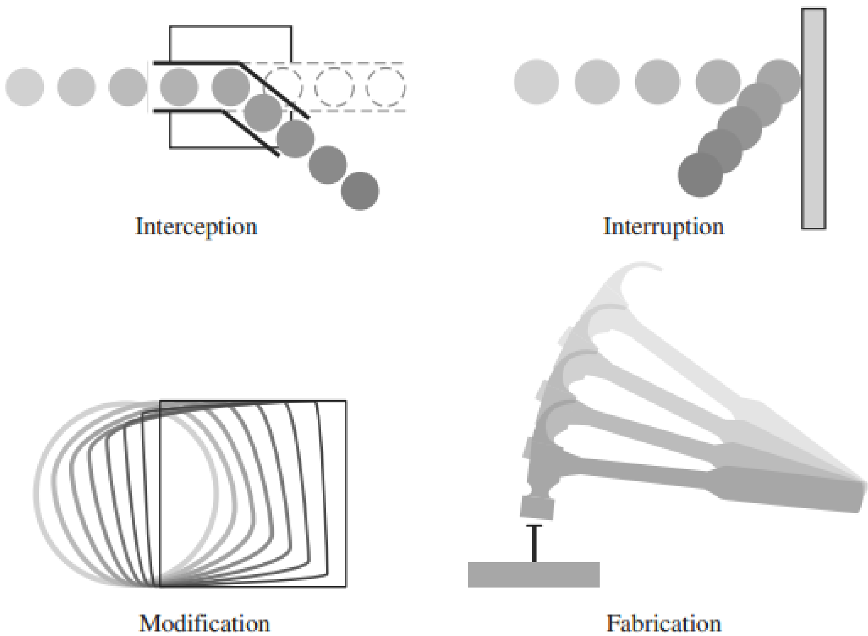


Figure 1: Four acts that cause security harm

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 2.1.
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 stakeholders' *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 [21] - systems have to be constantly updated and possibly upgraded.
A Security Architecture is therefore a union of security objectives by different stakeholders and, if properly implemented, guarantees the stakeholders' needs. These needs have to be considered throughout the system processes and have to be registered in a comprehensible manner. 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 [18]. 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.

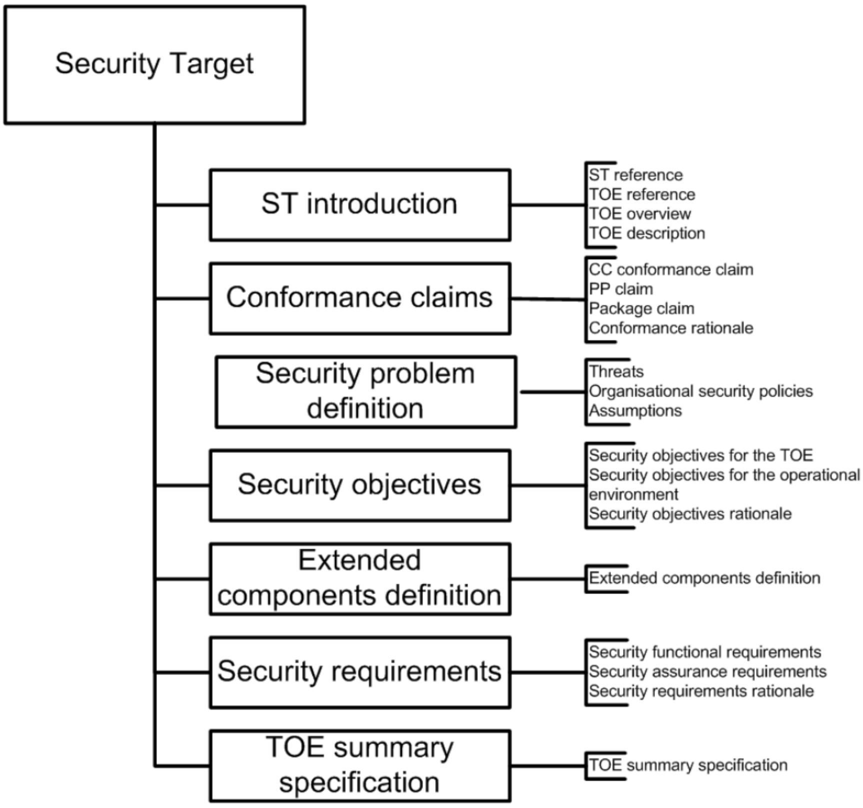


Figure 3: Overview of the Security Target contents

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 [17] 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 [17].
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.

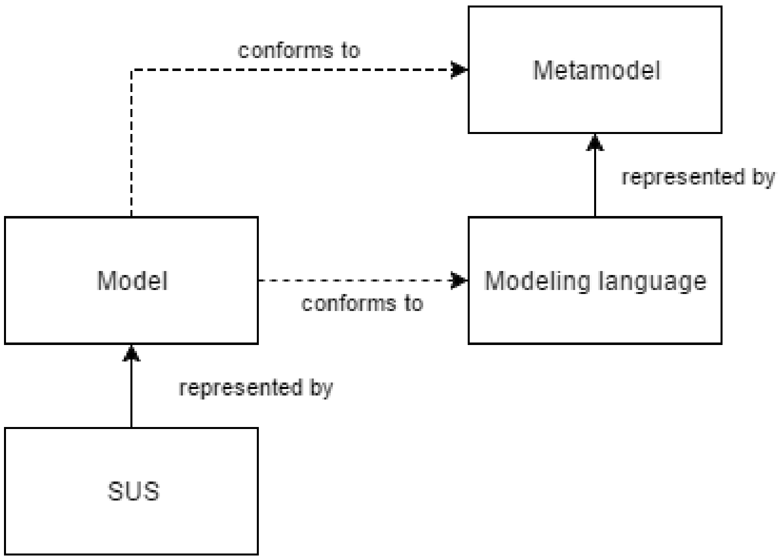


Figure 4: Relationships between Model and Metamodel

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) [13] 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. [5] which can be seen in Figure 5.

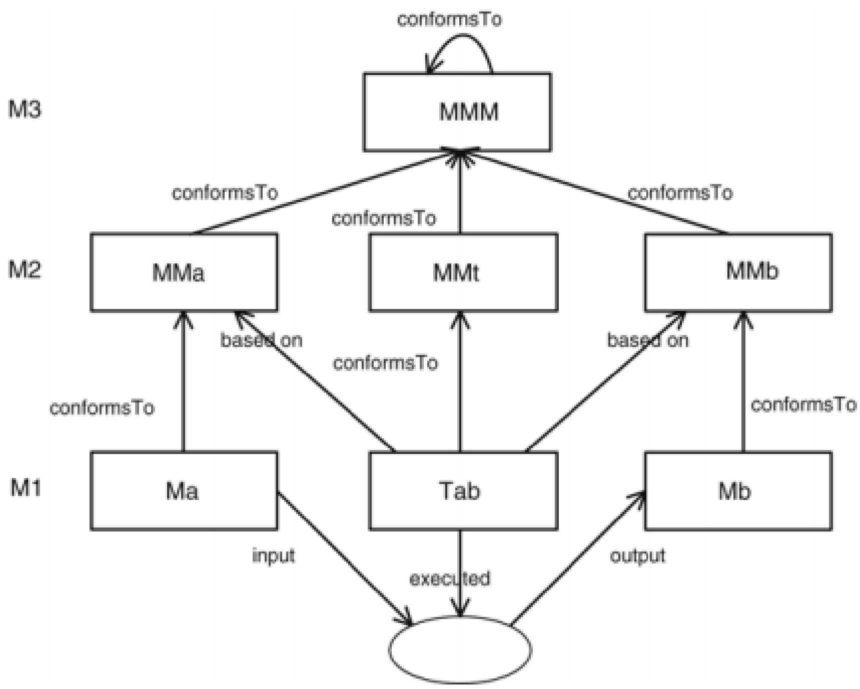


Figure 5: Model transformation according to Jouault et al.

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 [7].
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 [9].
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 identical. We can therefore simplify the transformation graph shown in Figure 5.

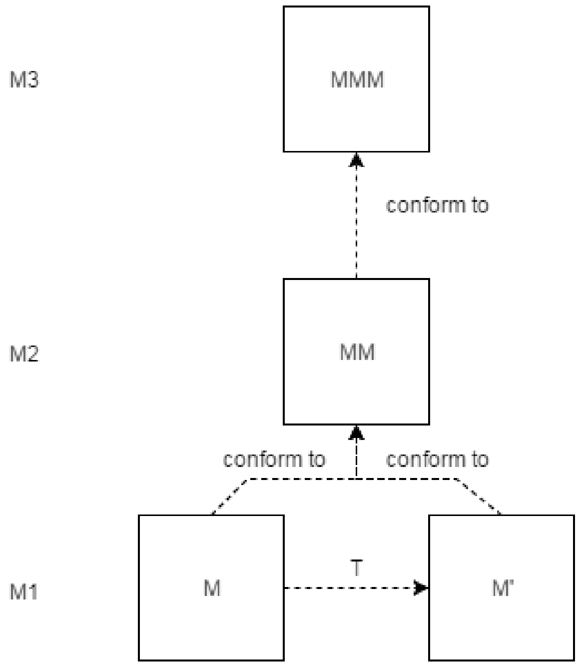


Figure 6: Model transformation

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]. Thyssen et al. differentiate between two possibilities, *Whole-part decomposition* which decomposes a system into smaller sub-systems 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 sub-systems 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.

3.1 Granularity layers

Thyssen et al. presented a framework which allowed them to view models under different levels of abstraction [19]. This abstraction is achieved by introducing granularity levels which are based on the different needs of the stakeholders.
Two different approaches were suggested where one is *Whole-part decomposition* which applies the divide-and-conquer principle and decomposes the SUS into smaller blocks until atomic blocks are reached. The other possible method is viewing the system from *Distinct development perspectives*. The differences have already been briefly discussed in Section 2.2.2 and therefore only whole-part decomposition will be presented in more detail.
Thyssen et al. propose a decomposition of systems into sub-systems to overcome the complexity. A sub-system is seen as a independent system by itself and the goal is to achieve „seamless“ abstraction. This way the sub-systems could be viewed separately and, if needed, could be aggregated to the overall system.

Different development perspectives were also proposed to focus on specific stakeholders and their needs during the system abstraction.
The *user perspective* describes, as the name suggests, the user's interests and the fulfillment of such by the respective system. The focus is on the hierarchical structuring of the general system functionality based on the users' needs and the functional interrelationships between components. The *logical perspective* can be seen as a bridge between the user-defined requirements and the technical implementation and lastly, the *technical perspective* describes the technical side, i.e. the hardware components that incorporate the needs of the respective users.
In this thesis however, the main focus is the combination of the security perspective and the „seamless“ abstraction of systems. As previously mentioned security in information systems is seen as a complex process [21] and the realization of security policies may have effects on different perspectives. Throughout the thesis system abstraction will be addressed from the perspective of a security architect that tries to incorporate all the stakeholders' security needs in a system. The goal is to propose a solution on how security properties of systems propagate through different levels of abstraction. Dependencies amongst them have to be taken into account and appropriate aggregation rules have to be defined.
Thyssen et al. have introduced the theoretical idea of „seamless“ abstraction but have not addressed the the transformation steps that are needed to transform a system from one granularity level to another.

3.2 Security Attribute Aggregation

Aggregation of security attributes has been studied and aggregation rules have been discussed by multiple researchers. Menzel et al. focused on the aggregation of security requirements and dependencies amongst them [10]. They defined interaction sets for requirements that classified the effects two requirements r_1 and r_2 might have on each other. According to [10] requirements could be independent, equivalent or conflicting, just to name a few.
Furthermore, when addressing the actual aggregation, Menzel et al. argued that to propose correct rules one has to look at two core aspects. Firstly one has to determine whether two requirements belong to the same class of security goals or not and secondly, the addressed entities and dependencies amongst the requirements have to be considered.
The transformation rules, which will be defined in Section 4, use this concept as the basis. Even though Menzel et al. have not addressed the needed transformation steps they provided core aspects that are essential for the aggregation of requirements. These will also be considered during application

for different security attributes.
Similarities can be found when looking at the work carried out by Noel et al. [12] which addresses attack graph aggregation.
Attack graphs are a representation of possible vulnerabilities in a network that can be exploited by attackers. Interactions and dependencies are represented by edges and for large networks such graphs may become very complex. To reduce the just mentioned complexity rules have been introduced that hierarchically aggregate graph elements.
Rules for exploits and security conditions, i.e. states that exist prior to or after an exploit, have been proposed. Exploits are being aggregated based on the attacker/victim machines where only exploits on the same machines are being aggregated into exploit sets.
Conditions are being aggregated into condition sets when they are either preconditions or postconditions of the same exploit. Later on, an aggregation based on machine level is being executed, i.e. conditions are being aggregated if they all occur on the respective machine. A machine is therefore a union of all its conditions.
Both the idea of aggregation based on same elements/machines as well as the idea of union sets consisting of conditions will be adapted in this thesis.
Lastly, risk aggregation is being covered. Lenstra et al. [8] have discussed both the qualitative as well as the quantitative risk analysis providing both the advantages and disadvantages of the respective method. Qualitative risk analysis assigns subjective values to risks or threats (such as High, Medium or Low) and these values then display the severity of the respective threat or risk. Qualitative approaches however, only provide very vague indications which might not be suitable for specific, more complex systems.
Quantitative approaches view the underlying events as distribution functions and derive the overall risk of an element from said functions. As mentioned by Lestra et al. an example could be the *Annual Loss Expectancy* for an event which can then be aggregated when viewing different events.
In our case quantitative aggregation is not necessary and might not even be possible in many cases since the given security concepts might be underspecified.
Our goal is to provide a solution on security attribute propagation for as many systems as possible applying qualitative aggregation for security attributes defined in the metamodel in Section 4.1. Although it is less precise than quantitative approaches [8] it suffices when considering the possibility of underspecification of security concepts and the overhead during aggregation steps when applying such an analysis.

3.3 Validation of Model Transformations

When proposing the model transformation one has to determine its correctness. Varro et al. define four properties that verify the correctness of a model transformation [21].

> **Syntactic correctness:** The generated model is a syntactically correct model instance (is conform to its metamodel)
>
> **Termination:** Model transformations must terminate
>
> **Uniqueness:** Model transformations must be deterministic

The fourth criterion is the *semantic correctness* which usually implies a semantic equivalence between source and target model. Here however, the model transformation is a projection, i.e. a possible gain/loss of information is possible.
Oracle functions serve the purpose of validating a model transformation output. The definition of such may become very difficult because of the complex nature of models as data structures [11]. Model comparison is described as one possible solution to validate a model transformation. Mottu et al. introduce a general oracle containing six different functions validating a given test case. Our primary goal however is the semantic gain of the projection and to verify this we have to define a very specific oracle in the security context. Model verification through testing is the most common form of validation [3] and thus, we will provide an implementation of the transformation rules and will validate its results by testing with sample models as inputs. The purpose of the testing process is the *detection of errors in the implementation*, the *completion of the specification* and lastly the *assessment of the result*, as mentioned by Fleurey et al.
The most important aspect is the result assessment. In our case the satisfaction highly depends on the respective user (e.g. security engineer) and we therefore have to define a criterion which reflects a successful model transformation of the initial security concept while being independent of the user. The approach will be an oracle based on model comparison and will be discussed in Section 5.2.

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 and each element of the metamodel will be put in the security context.
The second part will deal with the actual transformation rules. Sets that are important for the model transformation will be 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 restrictions. 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.

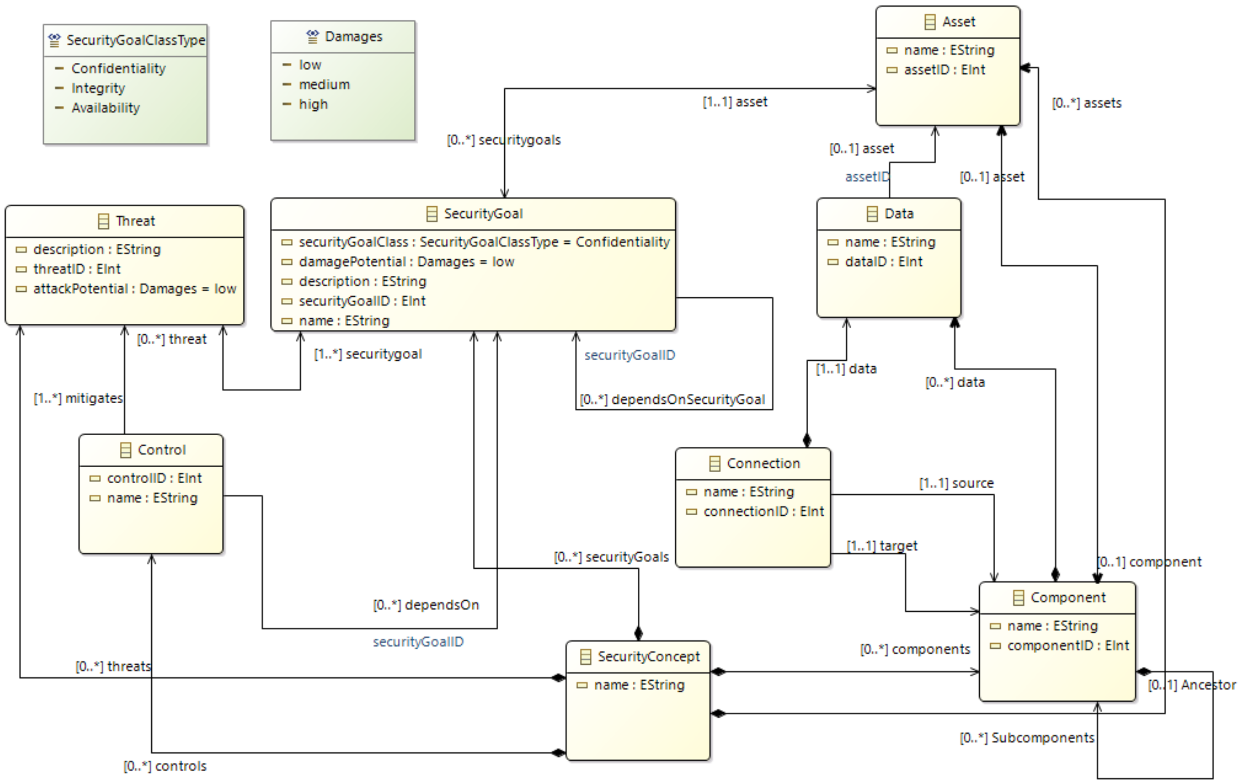


Figure 7: Security Concept Metamodel

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:

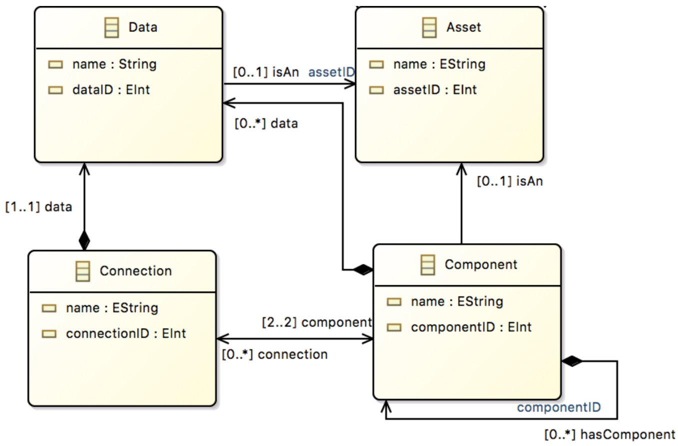


Figure 8: Two different interpretations of data

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.

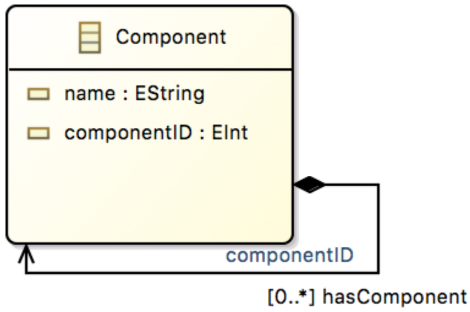


Figure 9: Structural information

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* [6]. 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 [16].

Data at Rest

Data at Rest applies to devices that hold data [6]. 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 *subComponent* 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. [10]. 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:

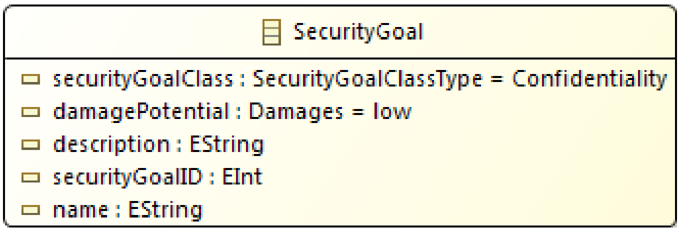


Figure 10: Attributes of a Security Goal

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.

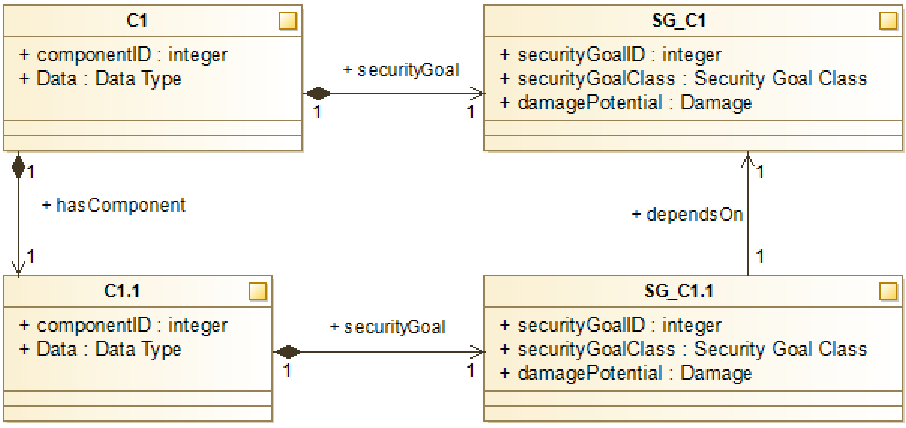


Figure 11: Relationships between Security Goals

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.

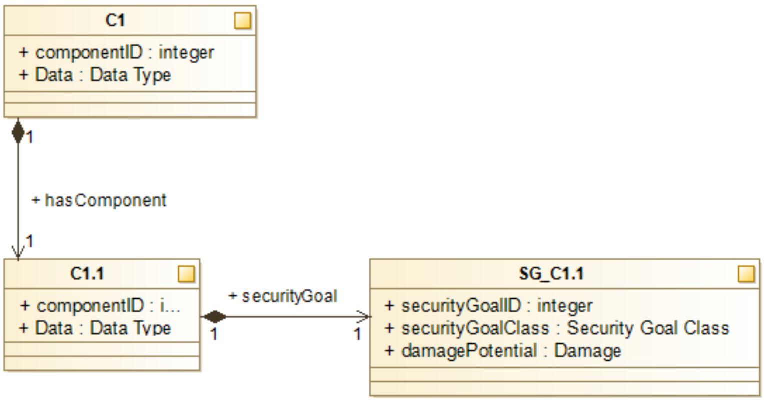


Figure 12: Ambiguous propagation

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_C1 of a component $C1$ has a direct influence on another component $C2$ 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 $C2$ processes/holds the data type that is being addressed by SG_C1*

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 exemplary 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.

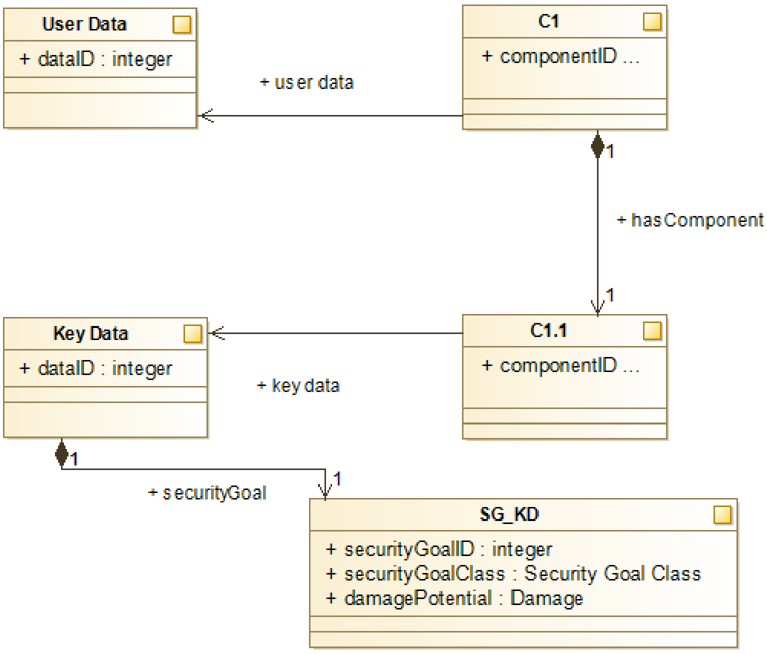


Figure 13: Ambiguous propagation

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 $C1.1$ 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 $C1.1$ to component $C1$ 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 $C1.1$ it is also protected in the layer above even though there is no explicit link between $C1$ and key data. This however, does *NOT* mean that the overall confidentiality is being protected since $C1$ 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 $C1$ is being protected then all of its sub-components are being protected as well.
For *Availability* however, the situation differs. The connection between components (*subComponent*) 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 $C1$ is protected it necessarily means that the availability of its sub-components is protected as well. To have the availability of a component protected all of the sub-components have to be protected individually. At the same time a component $C1$ may have different data or interconnections that are independent from their sub-components. We therefore cannot draw conclusions based on the availability of 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.

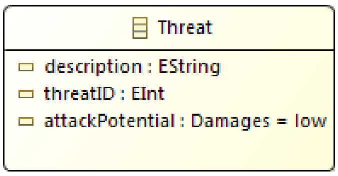


Figure 14: Attributes of a Threat

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.

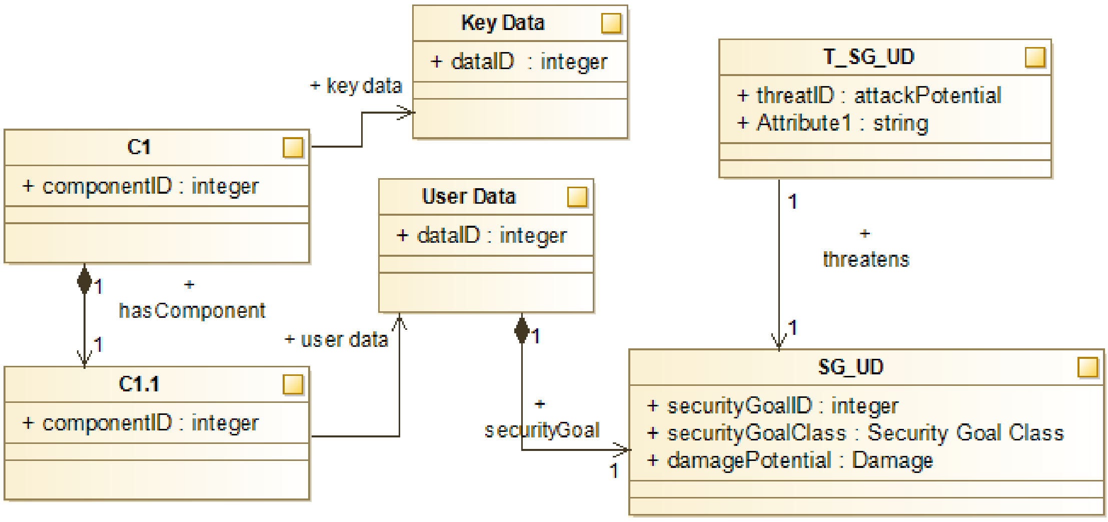


Figure 15: Underspecified Security Concept with a Threat

In Figure 15 a threat T_SG_UD threatens SG_UD which addresses the user data that is being used by sub-component $C1.1$. 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 ($C1$) to layers with higher granularity ($C1.1$) 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 $C1.1$ 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 integrity 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.

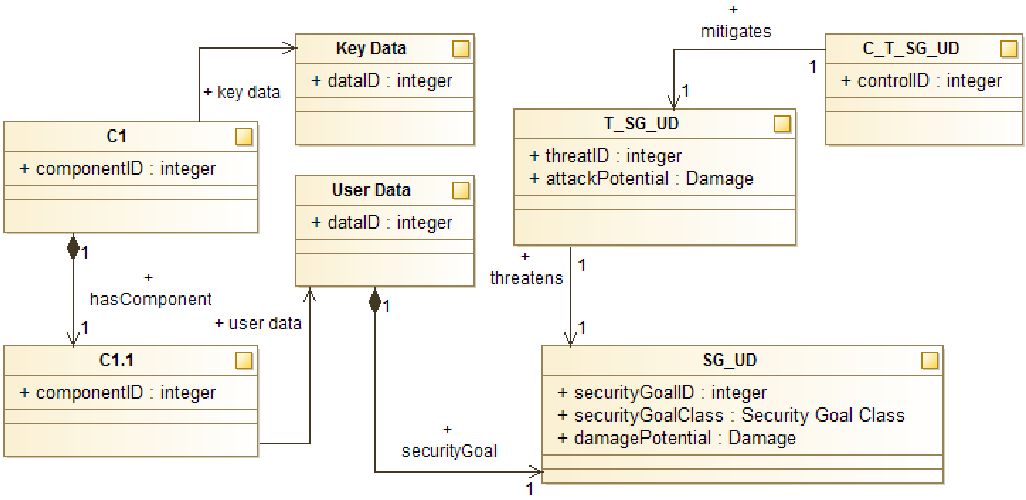


Figure 16: Underspecified Security Concept with a Control

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 $C1$ 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 $SG_{CIA}(SUS)$ be the set of security goals for a SUS. A security goal SG is defined as $SG(cl, c, asset, dmg)$ where cl is the security goal class (C for confidentiality, I for Integrity and A for availability), c being the component, $asset$ being the element of interest which the SG was defined for, i.e. the component itself or data which is being processed by it and dmg is the damage potential in case of a security breach (H = high, M = medium and L = low). Similarly we define a threat set T_{STRIDE} for a SUS where STRIDE is the threat modeling technique developed by Microsoft. According to [20] a component can be exposed to the following threats:

Spoofing: Adversaries pretend to be someone else

Tampering: Adversaries change data in transit

Repudiation: Adversaries perform actions that can't be traced back to them

Information disclosure: Unauthorized viewing/stealing of data

Denial of service: Interruption of system services

Elevation of privilege: Performing of unauthorized actions

Every threat has therefore a specific threat class that it can be assigned to. T_{STRIDE} is the set of threats where $T(tc, sg, sev)$ is a threat, tc being the threat class, sg being the security goal which the threat violates and sev being the severity of the threat if successfully exploited (H = high, M = medium and L = low).
Controls are tightly coupled with security goals and threats. C_{SUS} is the set of controls for a SUS that contains all the controls. A Control $C(T)$ has a set of threats T it mitigates.
Each asset can be viewed separately when looking at security goals and threats. For SGs we can define three separate sets, $SG_C(c)$, $SG_I(c)$ and $SG_A(c)$ where each one contains the respective SGs with either confidentiality, integrity or availability as the security goal class.

Definition 2 $SG_{CIA}(c) = SG_C(c) \cup SG_I(c) \cup SG_A(c)$

The respective sets covering one security goal class can be defined as follows:

Definition 3 $SG_{sgc}(c) = SG(sgc, c, asset_j, dmg) \cup SG(sgc, c, asset_{j+1}, dmg) \cup \ldots \cup SG(sgc, c, asset_m, dmg)$

Threats can be defined accordingly. For a threat T with the threat class tc the following union set can be defined.

Definition 4 $T_{tc}(c) = T(tc, sg_i, sev) \cup T(tc, sg_{i+1}, sev) \cup ... \cup T(tc, sg_n, sev) | sg_i \in SG_{CIA}(c)$

The set T_{STRIDE} is therefore a union of all threat sets:

Definition 5 $T_{STRIDE}(c) = T_S(c) \cup T_T(c) \cup T_R(c) \cup T_I(c) \cup T_D(c) \cup T_E(c)$

The just mentioned sets are the basic sets for components without looking at structural features of a SUS. As mentioned in Section 2.2 we can derive further security properties from sub-components and overall structural features of the system. This derivation will be covered in the following Sections.

4.4 Transformation Rule Set for Security Goals

The transformation rules for security goals will now be introduced. The following pseudocode will then be explained afterwards using an exemplary security concept. The function *compute_SG* has two inputs, *cid* is an ID of the component selected by the user and *security_concept* is the initial security concept model. The user-selected component IDs display the granularity level. The user can select as many components as needed and the result will be a security concept with the aggregated security attributes for the selected elements.

Algorithm 1 Transformation rules for security goals

security_concept ← security concept model
L_v ← list of visited nodes
L_c ← list of components of interest
L_{sg} ← list of security goals
S_{anc} ← ancestor node stack
S_{subc} ← children node stack

```
function COMPUTE_SG(cid, security_concept)
    L_sg ← empty
    component = findComponentByID(cid)
    if component not in L_v then
        L_v.add(component)
        for each sg in component.asset.securityGoals do
            L_sg.add(sg)
        end for
        for each con in component.connections do
            L_sg.add(con.data.asset.securityGoal)
        end for
        findAncestors(component)
        findChildren(component)
        checkConnections(L_sg, component)
        writeToSecurityConcept(cid, L_sg)
    else
        break
    end if
end function
function FINDANCESTORS(component, child)
    if component.Ancestor then
        if component.Ancestor in L_c then
            S_anc.add(component.Ancestor)
            findAncestors(component.Ancestor, child)
        end if
        add_sg_anc(component.Ancestor, component)
        findAncestor(component.Ancestor, child)
    else
        for each cmp in S_anc do
            sg_compute(cmp.id, security_concept)
            add_sg_anc(cmp.id, component)
        end for
    end if
end function
```

```
function findChildren(component, anc)
    if component.SubComponents then
        for each cmp in component.SubComponents do
            if cmp in L_c then
                S_subc.add(cmp)
                findChildren(cmp, anc)
                fixConnection(component, cmp)
            end if
            add_sg_child(cmp, component)
            fixConnection(component, cmp)
            findChildren(cmp, anc)
        end for
    else
        for each cmp in S_subc do
            sg_compute(cmp.id, security_concept)
            add_sg_child(cmp, component)
        end for
    end if
end function
function add_sg_AtoC(anc, child)                          ▷ ancestor to child
    for each asset in anc.assets do
        for each sg in anc.asset.securityGoals do
            if asset.componentID == anc.ids then
                sg.c = child
                sg.asset = child
                child.asset.securityGoals.add(sg)
            else if sg.securityGoalClass == 'Confidentiality'
              && sg.asset in child.assets then
                sg.c = child
                child.asset.securityGoals.add(sg)
            else if sg.securityGoalClass == 'Integrity'
              || 'Availability' then
                sg.c = child
                child.asset.securityGoal.add(sg)
            end if
        end for
    end for
end function
```

```
function ADD_SG_CTOA(child, anc)                                ▷ child to ancestor
    for each asset in child.assets do
        for each sg in child.asset.securityGoals do
            if !anc.assets.contain(child.asset) then
                copyAsset(child.asset, anc)
            else if sg.securityGoalClass == 'Confidentiality' then
                sg.c = anc
                anc.asset.securityGoals.add(sg)
            else if sg.securityGoalClass == 'Availability' then
                sg.c = anc
                anc.asset.securityGoal.add(sg)
            end if
        end for
    end for
end function
function FIXCONNECTION(child, anc)
    for each con in child.connections do
        if con.source == child then
            con.source = anc
        else if con.target == child then
            con.target = anc
        end if
    end for
end function
function CHECKCONNECTIONS(L_sg, component)
    for each con in component.connections do
        if (con.source || con.target) not in L_c then
            component.connections.delete(con)
        else
            L_sg.addEach(con.data.asset.securityGoals)
        end if
    end for
end function
```

4.4.1 Model Transformation using an Example

The following security concept serves as an example. The selected granularity level is displayed by the three marked components, $C1$, $C2$ and $C1.1.1$. Some of the components in the security concept have security goals defined.

```
function checkSecurityGoalDependencies(L_sg)
    for each sg in L_sg do
        for each dep_sg in sg.dependsOnSecurityGoal do
            if dep_sg.asset.component not in L_c then
                L_sg.delete(sg)
            end if
        end for
    end for
end function
```

The goal now is to derive the influence these goals have on the selected components. In this chapter a complete transformation will be carried out and the resulting security concept will be presented. Key transformation steps will be discussed and highlighted.

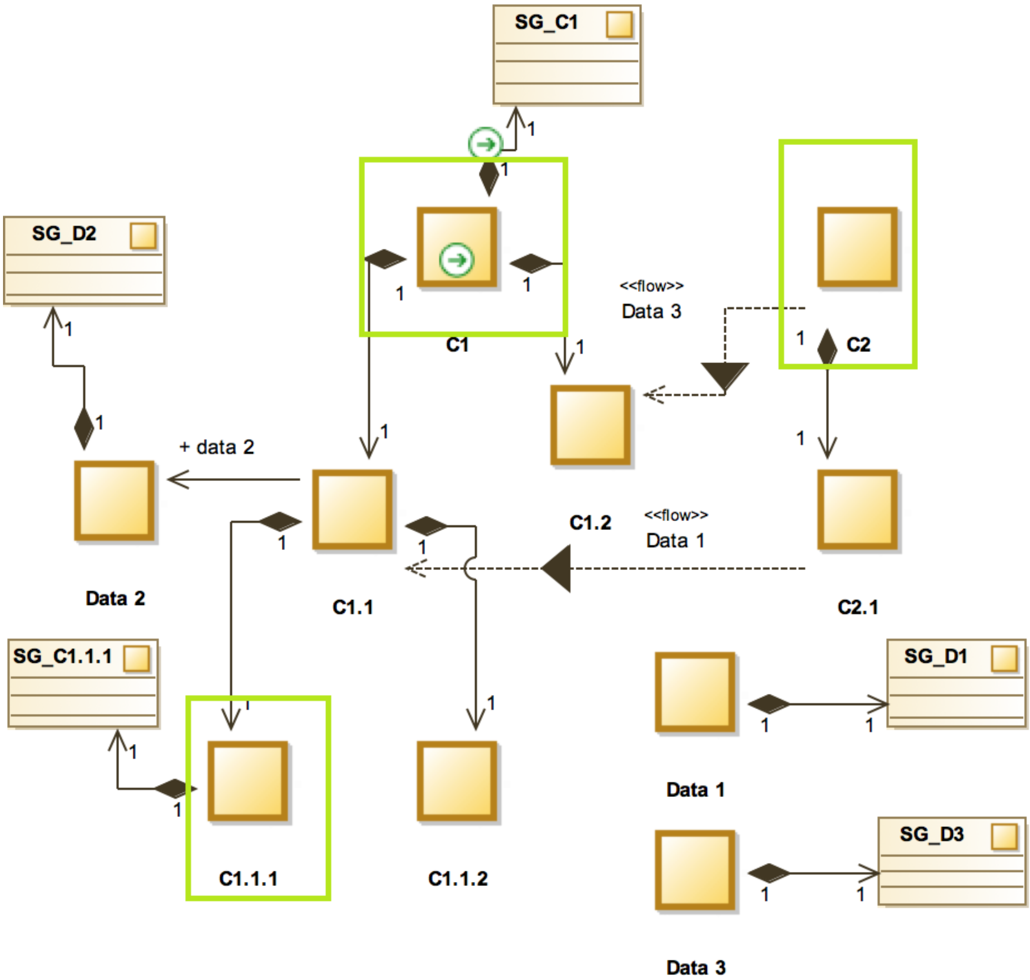


Figure 17: Security Concept Model

For every component of interest the function *compute_sg* will be called. First the list of components L_c will be populated with the three component s $C1$, $C2$ and $C1.1.1$. Prior to applying the algorithm, we have to define the security goals. A definition follows:

$$\begin{aligned} SG_{CIA}(sec\_concept) = \{ & SG(A, C1.1.1, C1.1.1, "high"), \\ & SG(C, C1.1, D2, "medium"), \\ & SG(I, C1, C1, "low"), \\ & SG(C, -, D1, "high"), \\ & SG(C, -, D3, "medium")\} \end{aligned}$$

To cover as many cases as possible we will start with component $C1.1.1$. The first step, as seen in line 6, is to add all the security goals of the component to its security goal list L_{sg}. We only have one goal to address and therefore the set will only have one element.

$$L_{sg} = \{SG(A, C1.1.1, C1.1.1, "high")\}$$

Now, we will have to look at the ancestors/children of the node. We differentiate between two kinds of ancestors, an ancestor that is in L_c and is therefore a component of interest and an ancestor that is not in L_c. Ancestors that are in L_c have to processed individually (line 30).
$C1.1$ is not in L_c and therefore only the function *add_sg_AtoC* will be called (line 54). SG_D2 is the only security goal for component $C1.1$ but since Data 2 is not an asset of the sub-component $C1.1.1$, it will not propagate to the abstraction layer below. $findAncestor$ is then called with the ancestor of $C1.1$, i.e. $C1$.
Since $C1$ is in L_c it will be added to the ancestor stack that the algorithm will have to work through. This is important in cases where there is a chain of components that are all in L_c. In such case, we would have to iterate through all of those nodes starting with the one in the highest abstraction layer. $C1$ does not have an ancestor and therefore the function *sg_compute* for $C1$ will be called (line 30).
Every time *sg_compute* is being called the respective component is being added to the list of visited nodes (L_v) to ensure that nodes would not be processed multiple times (line 5). After calling *sg_compute* with $C1$ as the component, our list is now:

$$L_v = \{C1.1.1, C1\}$$

Analogous to $C1.1.1$, the security goals of $C1$ will be added to L_{sg}.

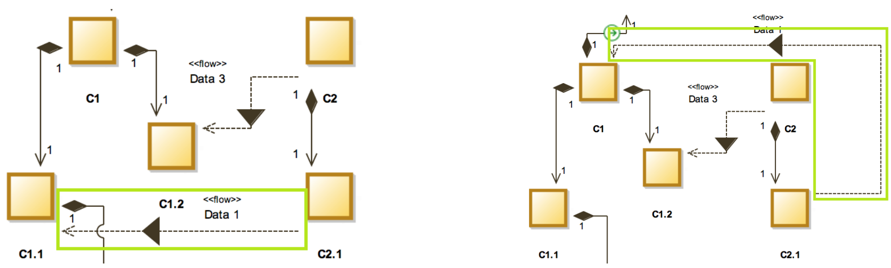
$$L_{sg} = \{SG(I, C1, C1, "low")\}$$

Since $C1$ does not have any ancestor nodes, the function $findChildren$ will be called. Every sub-component of the current component will be processed (line 37). Similar to the ancestor function we differentiate between two kinds of children, ones that are in L_c and ones that are not.
$C1$ has two sub-components $C1.1$ and $C1.2$ that have to be processed as well. We start with $C1.1$ which is not in L_c and therefore $C1.1$ will not be

added to the sub-component stack S_{subc}. The security goals however will be added to L_{sg} which still belongs to the ancestor $C.1$. The component must be altered. The initial security goal had $C1.1$ as the component which must be changed to $C1$. Moreover must one ensure that the asset exists in the layer above. This is not the case for $C1$ and therefore Data 2 will be copied and inserted into the abstraction layer above (line 77).

$$L_{sg} = \{SG(I, C1, C1, ''low''), SG(C, C1, D2, ''medium'')\}$$

After having the security goal added we will proceed with the sub-component of $C1.1$, namely $C1.1.1$. The function $find\_children$ is being called recursively with two parameters, the current component *component* which is $C1.1$ here and the initial parental node *anc*. This way the security attributes will be correctly aggregated regarding *anc*.
$C1.1.1$ has one security goal, $SG\_C1.1.1$ that addresses the availability of the component. Since there are no further children add_sg_CtoA will be called. According to the idea presented in Section 4.2.3 the protection of the availability of a sub-component does not necessarily mean the availability of the components in higher abstraction layers. Thus, $SG\_C1.1.1$ will not be added to L_{sg}. The other sub-component $C1.1.2$ does not possess any security attributes and does not have any influence on its ancestors.
The function $fixConnection$ that is called while processing the sub-components has not yet been addressed. The nodes $C1.1.1$ and $C1.1.2$ do not have connections to other components, $C1.1$ however, does. After iterating through the sub-components of $C1.1$ this function is being called.
The connection, that would be otherwise lost during abstraction, has to be processed. The connection that has $C1.1$ as the target will be changed in a way that $C1$ becomes the target. This way potential security attributes that address this connection and the data being transmitted on it will not be lost during the transformation. The following figure depicts the changed connection.



(a) Before connection adjustment (b) After connection adjustment

When processing the second sub-component of $C1$ we find another connection that needs to be addressed as well. The target is being changed from $C1.2$ to $C.1$ as the following figure suggests.

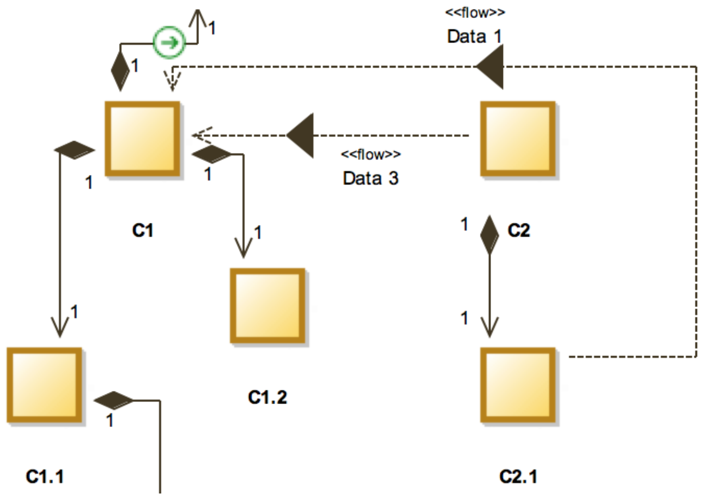


Figure 19: Connection adjustment

The function *checkConnections* will be called to verify that both sources and targets of the added connections are in L_c. Only then will the security attributes be added to the resulting model. In this case only one connection is correct having both the source and target in L_c. Therefore we will add the security goals addressing the transmitted data Data 3. The other connection will be deleted for now since the source is not in L_c.

$$SG_{CIA}(C1) = \{SG(I, C1, C1, \text{"}low\text{"}), SG(C, C1, D2, \text{"}medium\text{"}), \\ SG(C, C1, D3, \text{"}medium\text{"})\}$$

Lastly it is possible that security goals had been added to the sets even though the security goals they are dependent on were not. To verify the correctness of the transformation the function *checkSecurityGoalDependencies* will be called. Here however, no dependencies can be found.

The aggregation for $C1$ has been carried out while iterating through the ancestors of $C1.1.1$. To conclude the aggregation the function *add_sg_AtoC* will be called. $C1$ now has two security goals that might have an influence on $C1.1.1$. In Section 4.2.3 the propagation of security attributes through abstraction layers has been discussed. If the integrity for a component C is being protected it necessarily means that the integrity of its sub-components $C_i...C_n$ is being protected as well. We can therefore add SG_C1 to L_{sg} for

$C1.1.1$ while adjusting the properties of the security goal. The attributes component and assets will be changed from $C1$ to $C1.1.1$.

$$SG_{CIA}(C1.1.1) = \{SG(A, C1.1.1, C1.1.1, "high"), SG(I, C1.1.1, C1.1.1, "low")\}$$

There is one final component that has to be addressed. $C2$ is in L_c and similar to the previous nodes the function *compute_sg* will be called. $C2$ has no direct security attributes and no ancestors. The only component is the child $C2.1$ that has a connection to $C1$.
While processing $C2.1$ the function $fixConnection$ will be called that will put its ancestor as the source of the connection. The resulting model follows.

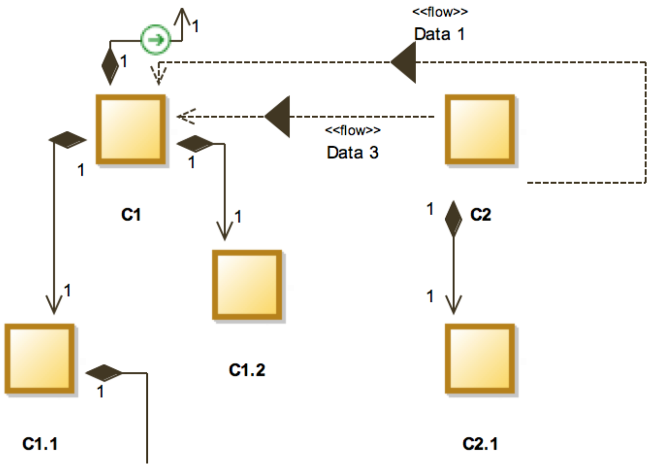


Figure 20: Connection adjustment

$checkConnections$ will be called at last and both outgoing connections will be processed. Since targets and sources are in L_c security attributes of both connections will be added to the security goal set of $C2$.

$$SG_{CIA}(C2) = \{SG(C, C2, D1, "high"), SG(C, C2, D3, "medium")\}$$

One more iteration is needed to assure that all connections of all components have been added correctly. As a result a security goal addressing Data 1 will be added to the security goal set of $C1$.

$$SG_{CIA}(C1) = \{SG(I, C1, C1, "low"), SG(C, C1, D2, "medium"),$$
$$SG(C, C1, D3, "medium"), SG(C, C1, D1, "high")\}$$

The transformation result as well as a comparison between the initial set of security attributes and the resulting set follows.

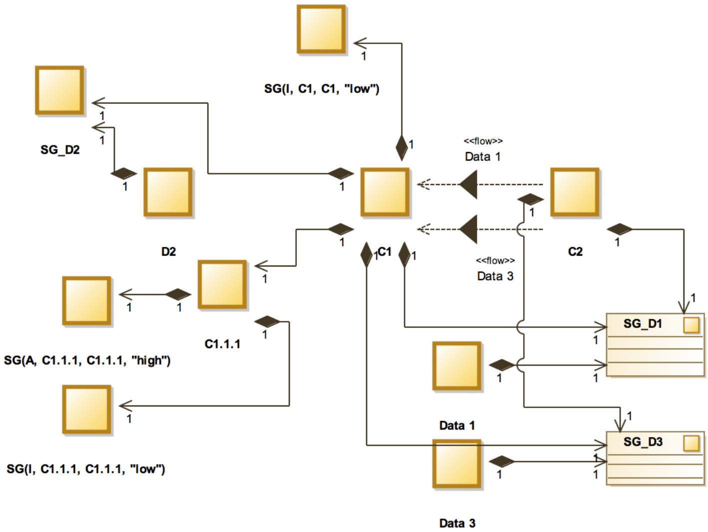


Figure 21: Security concept after the transformation

$$SG_{CIA}(C1) = \{SG(I, C1, C1, "low")\}$$
$$SG_{CIA}(C1.1.1) = \{SG(A, C1.1.1, C1.1.1, "high")\}$$
$$SG_{CIA}(C2) = \emptyset$$

Figure 22: Security goal set before aggregation

The initial security attribute sets for the respective components were sparse. Especially for component $C2$ where we cannot find any security attributes.

$$\begin{aligned} SG_{CIA}(C1) = \{&SG(I, C1, C1, "low"), SG(C, C1, D2, "medium"), \\ &SG(C, C1, D3, "medium"), SG(C, C1, D1, "high")\} \\ SG_{CIA}(C1.1.1) = \{&SG(A, C1.1.1, C1.1.1, "high"), SG(I, C1.1.1, C1.1.1, "low")\} \\ SG_{CIA}(C2) = \{&SG(C, C2, D1, "high"), SG(C, C2, D3, "medium")\} \end{aligned}$$

Figure 23: Security goal set after aggregation

The information gain that can be observed is substantial. We will now cover aggregation rules for security goals that can be used to derive new security goals based on the aggregated sets.

4.4.2 Aggregation Rules

So far we have only discussed the aggregation of security goals based on the granularity level of the SUS. The resulting security goal set has so far been a union of all the discovered security goals. As discussed in Section 3.2 there exist different approaches in how security attributes can be aggregated. In this thesis we will cover qualitative security attribute aggregation. In case of security goals the aggregation will be carried out based on the provided *damagePotential* in case of a breach. The following snippet displays the idea:

```
function securityGoalAggregation(L_sg)
    final_sgs ← empty
    for each sg in L_sg do
        tmp_sg = L_sg.where(_sg => _sg.securityGoalClass == sg.securityGoalClass && _sg.asset == sg.asset)
        tmp_sg.first.damagePotential = tmp_sg.where(damagePotential.max)
        final_sgs.add(tmp_sg.first)
    end for
end function
```

Let SG_{CIA} be the set of security goals for a component C_i. After iterating through the SUS and deriving security goals we receive the following set:

$$\begin{aligned} SG_{CIA}(C_i) = \{ & SG(I, C_i, C_i, \text{"}low\text{"}),\\ & SG(C, C_i, D_j, \text{"}medium\text{"}),\\ & SG(I, C_i, C_i, \text{"}high\text{"}),\\ & SG(C, C_i, D_j, \text{"}medium\text{"})\} \end{aligned}$$

The set has four security goals out of which two each address the same security goal class for the same asset. This might for example happen if two sub-components have security goals addressing their integrity and both propagate into the abstraction layers above up to C_i.

We have now two options. We either take this, possibly redundant, set as the result of the model transformation or we try to aggregate the security goals. When applying the *securityGoalAggregation* function we iterate through L_{sg} and try to find security goals with the same security goal class and asset. Having found a match, we take the maximum *damagePotential* and set this as the potential of the aggregated security goal. For this example the resulting set would be:

$$
\begin{aligned}
SG_{CIA}(C_i) = \{&SG(I, C_i, C_i, "high"),\\
&SG(C, C_i, D_j, "medium")\}
\end{aligned}
$$

This aggregation is optional and should be carried out by the user if needed. When aggregating security goals we also have to discuss the effect it has on the closely coupled attributes threats and controls. This effect will be highlighted in Sections 4.5.1 and 4.6.1.

4.5 Transformation Rule Set for Threats

When covering the aggregation of threats we will take the example shown in Section 4.4.1 and extend it by adding threats to the security concept. Further, to avoid redundancy it will be explained how to aggregate threats while iterating through components using the previously defined methods for security goals without providing code snippets.
As a start we have to define threats that threaten the initially defined security goals in Section 4.4. This was the initial set:

$$
\begin{aligned}
SG_{CIA}(sec\_concept) = \{&SG(A, C1.1.1, C1.1.1, "high"),\\
&SG(C, C1.1, D2, "medium"),\\
&SG(I, C1, C1, "low"),\\
&SG(C, -, D1, "high"),\\
&SG(C, -, D3, "medium")\}
\end{aligned}
$$

We will therefore define multiple threats addressing the just mentioned security goals. To put everything into perspective we will firstly add the „dummy “threat into our security concept to show the overall system picture.

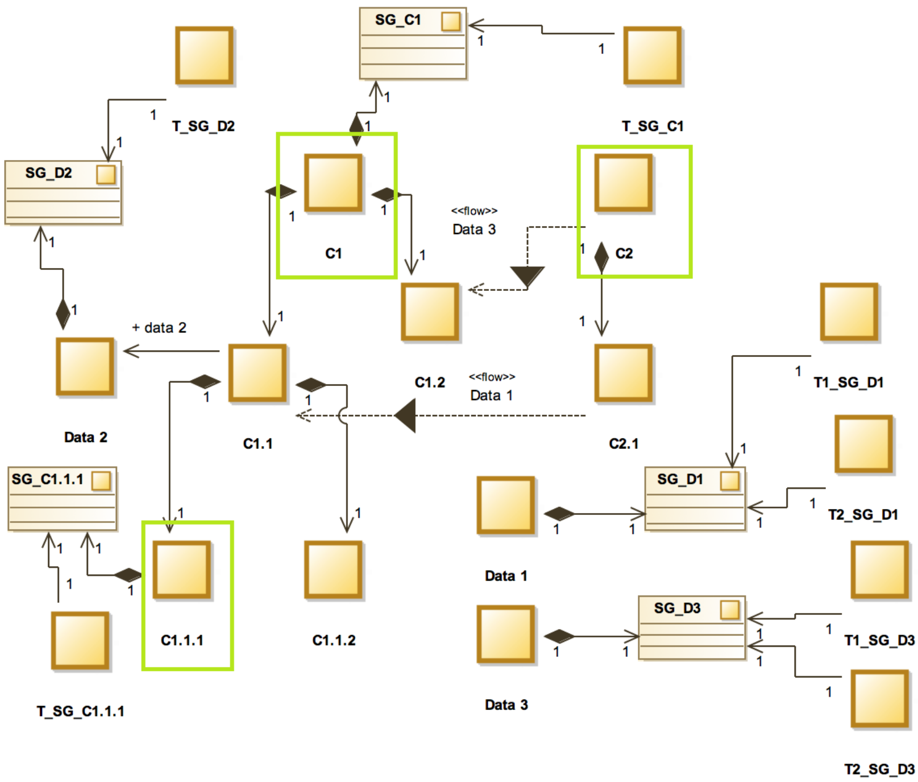


Figure 24: Security Concept Model

Now follows the definition of the respective threats with their attributes.

$$
\begin{aligned}
T_{STRIDE}(sec\_concept) = \{ & T(D, SG(A, C1.1.1, C1.1.1, \text{"}high\text{"}), \text{"}low\text{"}), \\
& T(T, SG(C, C1.1, D2, \text{"}medium\text{"}), \text{"}high\text{"}), \\
& T(S, SG(I, C1, C1, \text{"}low\text{"}), \text{"}medium\text{"}), \\
& T(I, SG(C, -, D1, \text{"}high\text{"}), \text{"}low\text{"}), \\
& T(I, SG(C, -, D1, \text{"}high\text{"}), \text{"}medium\text{"}), \\
& T(I, SG(C, -, D3, \text{"}medium\text{"}), \text{"}medium\text{"}), \\
& T(E, SG(C, -, D3, \text{"}medium\text{"}), \text{"}high\text{"})\}
\end{aligned}
$$

To start off the transformation we have to define a list of threats L_t where threats will be added to during the iteration. Similar to the previous section we begin with $C1.1.1$ and the threats that are associated with this component. We will add the threats on data and on the component itself. The only threat will be added to L_t.

$$L_t = \{T(D, SG(A, C1.1.1, C1.1.1, "high"), "low")\}$$

When looking through the ancestors the threats will be added exactly when the security goals of the ancestor is added to L_{sg}. For $C1.1.1$ the direct ancestor is $C1.1$ but because the security goal does not propagate to the layer below, neither will the associated threats. It is important to note that even though we differentiate between rules for security goals, threats and controls all of these steps happen during one single iteration.
The ancestor of $C1.1$ is $C1$ and since it is in the stack *sg_compute* is being called with $C1$ as the parameter. We will add $C1$ to L_v and the initial threats to L_t.
We can only find one threat for the security goal addressing the integrity of $C1$ and will add it to L_t.

$$L_t = \{T(S, SG(I, C1, C1, "low"), "medium")\}$$

When looking at the sub-components of $C1$, $C1.1$ and $C1.2$ all security goals will be added. In this case there is only one direct security goal which is addressing the confidentiality of Data 2. We will therefore add the corresponding threat to L_t. It is important to note that the added threat already addresses the altered security goal, i.e. the asset and component has to be changed accordingly.

$$\begin{aligned} L_t = \{&T(S, SG(I, C1, C1, "low"), "medium"), \\ &T(T, SG(C, C1, D2, "medium"), "high")\} \end{aligned}$$

The other sibling $C1.2$ has no security goals and therefore no threats that have to be considered. The big difference when looking at threats can be seen in the following step when changing to a lower abstraction layer and processing $C1.1.1$. The only security goal for this component addresses its availability and in the previous section we have discussed that it will not propagate to the higher abstraction layers because of the proposed interpretations in Section 2.2.
Threats however, behave differently. Each and every threat of a sub-component has a direct influence on its parental nodes. A component can thus be only as secure as its sub-components. In this case the threat threatening the availability of $C1.1.1$ will also threaten its ancestors. We therefore add the threat to L_t.

$$
\begin{aligned}
L_t = \{&T(S, SG(I, C1, C1, \text{"}low\text{"}), \text{"}medium\text{"}),\\
&T(T, SG(C, C1, D2, \text{"}medium\text{"}), \text{"}high\text{"}),\\
&T(D, SG(A, C1, C1, \text{"}high\text{"}), \text{"}low\text{"})\}
\end{aligned}
$$

The next step during the security goal aggregation is the adjustment of connections. Since threats are coupled with security goals we will add the respective threats when calling *checkConnections* in the end of the security goal aggregation for a component. Only when both the source and target of a connection are in L_c, the goal and the threats will be added to L_{sg} and L_t. As mentioned in the previous section the target of the connection was changed from $C1.1$ to $C1$, the source however has not been changed yet and is therefore not in L_c.

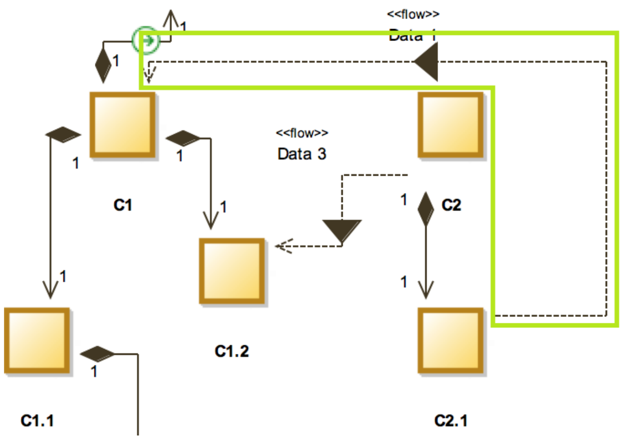


Figure 25: Connection adjustment

When looking at the sibling $C1.2$ we will find another connection that has to be adjusted as well. The source will be changed from $C1.2$ to $C1$ and the function *checkConnections* will be called.

Our components of interest are $C1.1.1$, $C1$ and $C2$, the source component of the connection transmitting Data 1 is $C2.1$ and as a result this connection will be deleted (line 100). The other connection has both the source and target in L_c and its associated security goals will be added to the set. Threats will be added accordingly.

```
function checkConnections(L_sg, component)
    for each con in component.connections do
        if (con.source || con.target) not in L_c then
            component.connections.delete(con)
        else
            L_sg.addEach(con.data.asset.securityGoals)
            L_t.addEach(con.data.asset.securityGoals.threats)
        end if
    end for
end function
```

The resulting threat set would be the following:

$$\begin{aligned} L_t = \{ & T(S, SG(I, C1, C1, "low"), "medium"), \\ & T(T, SG(C, C1, D2, "medium"), "high"), \\ & T(D, SG(A, C1, C1, "high"), "low"), \\ & T(I, SG(C, C1, D3, "medium"), "medium"), \\ & T(E, SG(C, C1, D3, "medium"), "high")\} \end{aligned}$$

Threat aggregatin for $C1$ was concluded for now and we will now look at $C1.1.1$ that might inherit threats from its parental nodes. As a reminder this was the set before the call of *compute_sg* for $C1$.

$$L_t = \{T(D, SG(A, C1.1.1, C1.1.1, "high"), "low")\}$$

The security goal addressing the integrity of $C1$ will be adapted and added to L_{sg} of $C1.1.1$. The reasons for this aggregation have already been discussed. When adding the security goal its threats will also be added to the resulting threat set.

$$\begin{aligned} T_{STRIDE}(C1.1.1) = \{ & T(D, SG(A, C1.1.1, C1.1.1, "high"), "low"), \\ & T(S, SG(I, C1.1.1, C1.1.1, "low"), "medium")\} \end{aligned}$$

Lastly we will look into the component $C2$ and its children. The connections will be adjusted once again and all the threats will be added to the resulting threat set of $C2$.

$$
\begin{aligned}
T_{STRIDE}(C2) = \{&T(I, SG(C, C2, D3, \text{"}medium\text{"}), \text{"}medium\text{"}),\\
&T(E, SG(C, C2, D3, \text{"}medium\text{"}), \text{"}high\text{"}),\\
&T(I, SG(C, C2, D1, \text{"}high\text{"}), \text{"}low\text{"}),\\
&T(I, SG(C, C2, D1, \text{"}high\text{"}), \text{"}medium\text{"})\}
\end{aligned}
$$

A function iterating through the connections will add the security goal addressing Data 1 to the component $C1$ since both source and target of the respective connection are in L_c. The threats will be added accordingly.

$$
\begin{aligned}
T_{STRIDE}(C1) = \{&T(S, SG(I, C1, C1, \text{"}low\text{"}), \text{"}medium\text{"}),\\
&T(T, SG(C, C1, D2, \text{"}medium\text{"}), \text{"}high\text{"}),\\
&T(D, SG(A, C1, C1, \text{"}high\text{"}), \text{"}low\text{"}),\\
&T(I, SG(C, C1, D3, \text{"}medium\text{"}), \text{"}medium\text{"}),\\
&T(E, SG(C, C1, D3, \text{"}medium\text{"}), \text{"}high\text{"}),\\
&T(I, SG(C, C1, D1, \text{"}high\text{"}), \text{"}low\text{"}),\\
&T(I, SG(C, C1, D1, \text{"}high\text{"}), \text{"}medium\text{"})\}
\end{aligned}
$$

Because of the model transformation and the rules for threat aggregation we were able to extend the number of threats. For component $C1$ the total number of threats is seven in comparison to the single initial threat.
Similarly to security goals we will now cover possible aggregation rules for threats.

4.5.1 Aggregation Rules

For security goals we have defined the following example security goal set:

$$
\begin{aligned}
SG_{CIA}(C_i) = \{&SG(I, C_i, C_i, \text{"}low\text{"}),\\
&SG(C, C_i, D_j, \text{"}medium\text{"}),\\
&SG(I, C_i, C_i, \text{"}high\text{"}),\\
&SG(C, C_i, D_j, \text{"}medium\text{"})\}
\end{aligned}
$$

The aggregation has been carried out according to the attribute $damagePotential$. As a result we have two security goals, each addressing one asset.

$$\begin{aligned}SG_{CIA}(C_i) = \{&SG(I, C_i, C_i, "high"),\\ &SG(C, C_i, D_j, "medium")\}\end{aligned}$$

During this aggregation however we have to also look into the associated threats that must also be aggregated.

---

function SECURITYGOALAGGREGATION(L_{sg})
  final_sgs ← empty
  for each sg in L_{sg} **do**
    tmp_sg = L_{sg}.where(_sg => _sg.securityGoalClass == sg.securityGoalClass && _sg.asset == sg.asset)
    tmp_sg.first.damagePotential = tmp_sg.where(damagePotential.max)
    tmp_sg.first.threat = tmp_sg.where(damagePotential.max).threat
    final_sgs.add(tmp_sg.first)
  end for
end function

To complete the example we will now define a threat set based on the initial security goals.

$$\begin{aligned}T_{STRIDE}(C_i) = \{&T(T, SG(I, C_i, C_i, "low"), "high"),\\ &T(I, SG(C, C_i, D_j, "medium"), "medium"),\\ &T(R, SG(I, C_i, C_i, "high"), "medium"),\\ &T(I, SG(C, C_i, D_j, "medium"), "low")\}\end{aligned}$$

After applying the function for security goal aggregation, we receive the following set:

$$\begin{aligned}T_{STRIDE}(C_i) = \{&T(R, SG(I, C_i, C_i, "high"), "medium"),\\ &T(I, SG(C, C_i, D_j, "medium"), "low")\}\end{aligned}$$

In this example the threat with the lower *attackPotential* has been chosen. The reason herefore is the corresponding security goal. Even though the attack potential is higher, the *damagePotential* in case of a security breach is lower. We therefore indirectly choose the threat based on the damage potential of the corresponding security goal.

4.6 Transformation Rule Set for Controls

Transformation rules for controls will only be covered very briefly. Every time when threats are being added to the respective threat set L_t of an element C_i the corresponding controls mitigating the threat are being added as well. An example showcasing this is unnecessary here.

4.6.1 Aggregation Rules

Aggregation for controls is very similar to the previous aggregations. Whenever a threat is being aggregated its controls have to be processed as well.

```
function securityGoalAggregation(L_sg)
    final_sgs ← empty
    for each sg in L_sg do
        tmp_sg = L_sg.where(_sg => _sg.securityGoalClass ==
sg.securityGoalClass && _sg.asset == sg.asset)
        tmp_sg.first.damagePotential = tmp_sg.where(damagePotential.max)
        tmp_sg.first.threat = tmp_sg.where(damagePotential.max).threat
        tmp_sg.first.threat.control = tmp_sg.where(damagePotential.max)
.threat.controls
        final_sgs.add(tmp_sg.first)
    end for
end function
```

This code snippet is self-explanatory. All controls mitigating the aggregated threat are being added to the resulting security concept.

5 Implementation

In this section the chosen technologies that has been used to implement the just mentioned transformation rules are being covered.

5.1 Technology

The implementation can be viewed from two perspectives, the modeling perspective and the actual implementation. Approaches for both perspectives will be introduced.

The *Eclipse Modeling Framework* (EMF) was used to create the security concept metamodel. EMF allows users to create their own models and

to generate Java code based on the modeled dependencies. Classes such as Threats, Controls and Security Goals were defined as well as datatypes such as *Damage* or *SecurityGoalClass*. The modeling itself was done using *EcoreTools* [1] that allows a user interact with the EMF using Eclipse, an Integrated Development Environment (IDE).
The automatically generated code can be used to interact with the metamodel and its elements. *Xtend* [2] was chosen as the language for the implementation of the transformation rules. *Xtend* is a Java dialect that is being compiled to Java code. It can therefore be seamlessly integrated into the EMF environment and can interact with the auto-generated code by the EMF framework.

5.2 Validity & Verification

As mentioned in Section 5.2 model verification through testing is the most common form of validation [3].
To be able to verify whether or not the model transformation achieved its goal of providing new information for the respective user we will have to create an oracle based on model comparison. Even though the security interpretation of a concept is highly dependent on the user we will try to create a criterion based on which we will be able to determine whether the transformation was a success or not.
Mottu et al. [11] proposed six oracle functions that checked the validity of a model transformation output. Here, the main focus to provide a function that is able to determine the information gain from a security perspective. A possibility would be to iterate through the components of the resulting set and compare their security attributes with the initial set.

[1] EcoreTools, online:`http://www.eclipse.org/ecoretools/index.html`
[2] Xtend, online: `http://www.eclipse.org/xtend/`

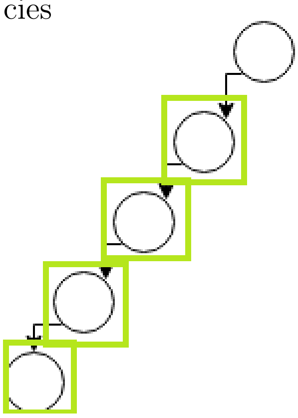
```
function securityConceptValidation(sc_old, sc_new)
    for each comp in sc_new do
        for each asset in comp.assets do
            _comp = findComponentById(sc_old, comp.id)
            if _comp.asset.securityGoals.size >
              comp.asset.securityGoals.size ||
             _comp.asset.securityGoals.threats.size >
             comp.asset.securityGoals.threats.size ||
             _comp.asset.securityGoals.threats.controls.size >
             comp.asset.securityGoals.threats.controls.size then return false
            end if
        end for
    end for
end function
```

It is a very trivial oracle that only compares the number of security attributes of the resulting model with the initial set of security attributes. If the set after the model transformation is equal or bigger than the initial set then the transformation was successful. The validation function has to be called prior to *securityGoalAggregation* because it is possible that security goals might be united with others and be therefore non-existent in the form they were in the initial set.

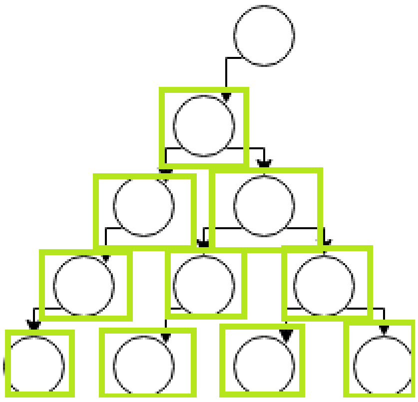
5.3 Runtime & Complexity

- provide two „extreme“examples

One unbalanced tree with nodes of interest, testing recursion and dependencies



One security concept with multiple hundred of nodes where every node is a node of interest.



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] F. Fleurey, J. Steel, and B. Baudry. Validation in model-driven engineering: testing model transformations. In *Proceedings. 2004 First International Workshop on Model, Design and Validation, 2004.*, pages 29–40, Nov 2004.

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

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

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

[7] 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.

[8] Arjen Lenstra and Tim Voss. *Information Security Risk Assessment, Aggregation, and Mitigation*, pages 391–401. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[9] 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).

[10] 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.

[11] J. M. Mottu, B. Baudry, and Y. L. Traon. Model transformation testing: oracle issue. In *2008 IEEE International Conference on Software Testing Verification and Validation Workshop*, pages 105–112, April 2008.

[12] 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.

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

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

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

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

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

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

[19] 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.

[20] P. Torr. Demystifying the threat modeling process. *IEEE Security Privacy*, 3(5):66–70, Sept 2005.

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