A Probabilistic Relational Model for Security Risk Analysis

Teodor Sommestad[a], Mathias Ekstedt [b] Pontus Johnson [c]

[a] Royal Institute of Technology (KTH), Industrial information & control systems, Osquldas väg 12, 7tr, 100 44 Stockholm, Sweden, Telephone: +46-8-7906920, Fax: +46-8-7906839, email: teodors@ics.kth.se

[b] Royal Institute of Technology (KTH), Industrial information & control systems, Osquldas väg 12, 7tr, 100 44 Stockholm, Sweden, email: mathiase@ics.kth.se

[c] Royal Institute of Technology (KTH), Industrial information & control systems, Osquldas väg 12, 7tr, 100 44 Stockholm, Sweden, email: pj101@ics.kth.se

Abstract

Information system security risk, defined as the product of the monetary losses associated with security incidents and the probability that they occur, is a suitable decision criterion when considering different information system architectures. This paper describes how probabilistic relational models can be used to specify architecture metamodels so that security risk can be inferred from metamodel-instantiations.

A probabilistic relational model contains classes, attributes, and class-relationships. It can be used to specify architectural metamodels similar to class diagrams in the Unified Modeling Language. In addition, a probabilistic relational model makes it possible to associate a probabilistic dependency model to the attributes of classes in the architectural metamodel. This paper proposes a set of abstract classes that can be used to create probabilistic relational models so that they enable inference of security risk from instantiated architecture models. If an architecture metamodel is created by specializing the abstract classes proposed in this paper, the instantiations of the metamodel will generate a probabilistic dependency model that can be used to calculate the security risk associated with these instantiations. The abstract classes make it possible to derive the dependency model and calculate security risk from an instance model that only specifies assets and their relationships to each other. Hence, the person instantiating the architecture metamodel is not required to assess complex security attributes to quantify security risk using the instance model.

Keywords: Security risk; risk assessment; architecture metamodel; probabilistic relational model; architecture analysis

1 Introduction

Security issues related to information technology continue to be a concern in today's society, and for decision makers in it. Security is a complex property, and several diverse factors need to be considered to assess the security of a system's architecture. To support decision makers a plethora of approaches, frameworks and methods has been proposed for analyzing and ranking security – all with some explicit or implicit definition of security.

From a decision maker's perspective, tools and techniques to assess security of both existing and potential future architectures are needed. There is also a need to relate the result of such an assessment to business decisions, such as investment alternatives that strengthen security. The concept of risk, defined as the product of the monetary losses associated with security incidents and the probability that they occur, has been suggested as a suitable input to decision making [1,2]. Several financial methods with risk measurements as a basis have also been adapted for security to provide decision makers with tools to manage security efficiently from a business perspective. For example return on security investment [3], and the methods presented in [4-7].

Although risk is well defined and practical for decision making, it is often difficult to calculate a priori. Analysis frameworks such as [4] restrict themselves to three variables: the probability that a threat surfaces, the probability that an attack succeeds, and the loss suffered from a successful attack. While quantifying these variables provides the necessary means for assessing risk, it is not apparent how to obtain the numbers needed to do so. Decision makers typically have an understanding of the architecture of their organization and its systems. However, their understanding of the dependencies among the properties of risk treatments, the threat

environment and sensitive assets is hazy. Methods that support decision makers by deriving security risk associated with both existing and potential future architectures are thus desirable.

Architectural models provide decision makers with a convenient tool to abstract and capture different aspects of information systems in diagrammatic descriptions. Metamodels like the one offered in CORAS [8] guide the modeler to create graphical descriptions that can be used to assess risk. This type of metamodels does however not help the modeler to identify the risks which their particular architecture face, and do not provide the data needed to quantify security or risk based on the model. This analysis is instead left for the user of the metamodel. Methods such as [9] generate attack graphs from descriptions of computer networks and offer an alternative when the decision concerns network security. But these do not provide support for assessing the probability that a certain threat surfaces, i.e. that certain attack steps are attempted, nor do they include losses in the models. Consequently, they do not produce a measure of security risk for the decision maker. This paper describes a formalism for constructing architecture metamodels so that security risk can be inferred from the metamodel's instantiations.

1.1 Architecture models and security risk analysis

If security risk could be easily quantified from architecture models of information systems this would provide an intuitive way to assess the security risk associated with both the current "as-is" scenario, and potential future "to-be" scenarios. The decision maker would create models of different architectures by representing relevant objects and relationships in diagrammatic descriptions and from these assess the security risk associated with the architectures. These architecture models may cover management aspects, operational aspects or pure technical aspects. They can for instance be created to assess the security risk associated with different network architectures, or to assess the impact of different password policies on the overall security risk.

To make accurate predictions from an architecture model it needs to represent objects and relationships that influence security risk. If network architectures are assessed, it would for example be of relevance to include information on the placement of firewalls in the architecture model. A metamodel can guide the decision maker to create instance models that include relevant objects and relationships. To provide this guidance the metamodel must resolve how security risks (according to some theory) depend on different architectures, at least to some level of detail.

If the security risk was possible to compute based on the theory of how risk relates to different architectures it would relieve the decision maker of extensive analytical efforts. Security risk could then be derived from instantiated architecture models, and the decision maker would only be required to represent the objects and relationships that constitute the architecture.

This paper proposes the use of probabilistic relational models (PRMs) [10] to specify metamodels for security risk analysis. A PRM is similar to a Class Diagram in the Unified Modeling Language (UML) [11] and contains classes, attributes, and class-relationships. In addition, a PRM makes it possible to associate a probabilistic model to the metamodel by defining relationships between the attributes of classes in the metamodel. More specifically, a PRM makes it possible to define how the value of one attribute depends on the value of other attributes in an architectural model. With these elements a PRM allows, in a general sense, architecture metamodels to be coupled to a probabilistic inference engine. A PRM can for instance specify how different logical network architectures and properties of its users influence the security risk an organization faces. Hence, if metamodels are expressed using the PRM formalism it can be specified how security risk should be inferred from the metamodel's instantiations.

There is however an infinite number of (more or less suitable) ways that a PRM can be structured for security risk analysis. A number of concepts need to be related to each other when security risk is assessed. The main contribution of this paper is the proposition of a package of abstract PRM-classes that can be used to create PRMs that infer security risk from architecture models.

The proposed class-package is expressed as a PRM and specifies a set of classes, attributes, class-relationships and a probabilistic model for how attributes of these classes depend on each other. The classes in this PRM are abstract and cannot be directly instantiated into an architecture model. They can however be made concrete if they are specialized into subclasses according to a set of constraints. If architecture models are instantiations of such concrete classes, then security risk is possible to infer from the architecture model. This inference can also be performed on architecture models that merely represent assets and assets relationships to each other. Hence, little security expertise is required to instantiate the architecture model, and security risk can still be inferred.

1.2 Outline

Chapter two describes related work within the field of security and risk analysis. Chapter three explains the PRM formalism and the terminology associated with it. Chapter four describes the relationship between the different models presented in subsequent chapters. Chapter five presents the main contribution of this paper – a PRM consisting of abstract classes that are associated with a set of constraints that state how these can be specialized into concrete subclasses. Chapter six exemplifies how these abstract classes can be specialized into concrete classes and how probabilistic models can be associated with these classes. In chapter seven a case study applying these specialized (concrete) classes to assess security risk associated with an automation system in power station is described. In chapter eight the proposed modeling method is discussed, and in chapter nine conclusions are drawn.

2 Related works

The use of architectural modeling languages has a long history in management and development of information systems. Modeling languages such as the Unified Modeling Language (UML) [11], the Systems Modeling Language (SySML) [12], and the Business Process Modeling Notation (BPMN) [13] provide support to create diagrammatic descriptions of information system architectures and system environments. These diagrammatic architecture descriptions can be developed for a variety of purposes, including different types of analysis. One aspect that can be analyzed based on architectural descriptions is security, and there are several methods and modeling languages specifically supporting this purpose. The formalism presented in this paper supports quantification of security risk based on system architecture models, and does not require security expertise to perform the actual quantification. This section will explain how similar modeling languages and methods relate to the one proposed in this paper.

There are several modeling languages targeted at providing support for security assessments in early phases of system development. Methods like misuse cases [14] and abuse cases [15, 16] align with well established methods for software requirements engineering to provide support for depicting potential threats and use cases that mitigate these. While these two offers a language to describe threats and countermeasures, they are not associated with methods to quantitatively assess security from the models created. This is also the case for languages such as SecureUML [17] and SPML [18] that use models with the purpose of model-driven development.

Two other modeling languages that are intended for the system development phases are Secure Tropos [19] and UMLsec [20]. Both of these provide a language and methodology which can provide a basis for security assessments. Secure Tropos extends the Tropos methodology [52] and can be used to specify security concerns associated with planned systems. The UML extension UMLsec provides a language to depict security-relevant information in diagrams describing a system specification [20]. Both these also provide support for automated verification of architecture models. Secure Tropos is associated with a set of rules that can be used to verify if goals related to trust are fulfilled when trust is delegated [50]. UMLsec makes it possible to evaluate if UMLsec-diagram fulfill a set of stipulated requirements [20,51]. The output of these automated analysis methods are however a pass/fail result that state if the architecture fulfill the requirements. Such verifications can support security risk analysis, but there are no automated means to compute security risk directly from them.

The pass/fail output is also a characteristic of the Common Criteria (CC) [21]. The CC framework offers a method to specify security requirements and evaluate their fulfillment. In CC's general conceptual model the relationships between owners, assets, risks, countermeasures, threat agents and threats are described. However, an evaluation employing CC's framework does not quantify risk. Instead it provides a pass/fail result together with a rating of the assurance level (1-7) of the evaluation.

A method specifically developed for analyzing and quantifying risk is CORAS [8]. With guidance from CORAS's metamodel, a graphical description of the threat scenario is created and used as a support to determine if, and how, the identified risks should be treated. This is done by modeling the relationships between assets, threats, vulnerabilities, unwanted events, risks and treatments. Although risk in CORAS is defined as the product of likelihood and consequence, there is no analysis framework coupled to the metamodel and thus no algorithmic method to calculate risk based on a graphical description. There is also no description of what different types of risk treatments that should be modeled, or how risk treatments influence risks in CORAS. These calculations, as well as the content of the CORAS diagram, must instead be assessed by the persons applying CORAS.

Several other analysis methods also depend on analysts to quantify risk. CCTA Risk Analysis and Management Method (CRAMM) [22] offer a structured method to assess risk qualitatively by identifying: 1) how frequent an incident occurs, 2) the probability that incidents would result in a worst case scenario, and 3) loss values. These

three values are used to produce a monetary value for annual loss expectancy. Information Security Risk Analysis Method (ISRAM) [23] does in a similar way guide the analyst to assess probabilities for security incidents to occur and to assess the potential consequences of these. The same type of guidance is also provided by Operationally Critical Threat, Asset and Vulnerability Evaluation (OCTAVE) [24].

Threat trees [25] and attack trees [26,27] are graphical notations that have evolved from fault trees, used to illustrate attackers' goals together with possible ways to reach these goals. The attacker's main goal is depicted as the root of the tree and the steps to reach this goal are broken down into sub-goals of the attack through "AND" and "OR" relationships. Threat trees and attack trees have been applied in several ways to assess security. In [25] it is suggested that the threat trees should be used to rank the threats is terms of risk. In [27] it is suggested that attack trees can be used to assess if attacks are possible, if special tools are needed or how much effort an attack requires.

A similar method for representing attacks is attack graphs. In attack graphs the sequence of steps needed to accomplish the attack is expressed rather than the set of steps; and this is modeled in a graph structure instead of a tree structure. Attack graphs have for instance been used to assess the probability that an attacker reaches particular attack step [28], or to analyze the security of system configurations in terms of the weakest adversary that can compromise the network [29]. Several similar analysis methods exist in addition to these (see for example [30-33]). A problem with attack graphs is that they scale poorly and become extremely complex even for moderately sized system architectures. Methods have been proposed to reduce the complexity and computational problems associated with attack graphs in [30,31,34]. Probabilistic treatment of the relationship between different attack steps is another suggested solution to the scalability problem. In [35] Bayesian networks are used to represent attack graphs more compactly and to calculate the probability that a network-attack succeeds, as oppose to doing so deterministically.

The methods based on trees and graphs provide a link between vulnerabilities and plausible consequences. Their structure also facilitates straightforward means for analysis, for example of how likely an how likely an attack is to succeed or if an attack step is reachable. None of these abovementioned methods does however offer means for assessing how likely an adversary is to attempt combinations of attacks steps. Hence, the probability of a certain attack being realized cannot be inferred and security risk cannot be calculated.

A natural extension of attack trees and attack graphs is to include controllable countermeasures in the model. In [25] countermeasures are added as leaves of threat trees; in [36] and [37] it is shown how threats and countermeasures can be related to each other in tree structures; in [26] countermeasures and attacks are connected in directed acyclic graphs. In [38] attack trees with countermeasures as leaves are called Defense trees. Techniques have been presented which use Defense trees to model strategic games in security [39], to model conditional preference of defense techniques using conditional preference nets [40], and for performing economic evaluation of security investments [38].

The method described in [38] allows a modeler to specify a defense tree and a number of variable values. From this, return on security investment as well as the adversary's return on attack can be inferred. As a step in these calculations annual loss expectancy is derived. There is however no method for deriving the defense tree or the variable values required from an architectural description, and new variable values must be defined after an option is chosen. Hence, the method relies on analysts to continuously update the model.

Methods have been developed to ease the burden of creating tree and graph structures by deriving them from architectural models. In [41-43] methods are described for deriving attack graphs from network configurations and known vulnerabilities; in [44] a method for deriving probabilistic defense graphs from architectural models is described. These methods do however focus strictly on prevention of attempted attacks and lack the power to represent treatments that limit the losses from successful attacks or deter adversaries from attempting them. Consequently, they do not provide the information required to determine security risk. The methods presented in [41-43] are also focused entirely on computer networks and thus lack the capability to represent other facets of security, such as human behavior or organizational policies.

Unlike the abovementioned approaches the formalism presented herein makes it possible to specify how expected loss should be quantified from an architecture model. The formalism makes it possible to specify architectural metamodels that generates a probabilistic dependency model when they are instantiated. The conceptual structure of this dependency model extends the conceptual model presented in Common Criteria [21] by defining attack steps as a part of a threat. Attack steps are related to countermeasures in directed acyclic graphs, similar to the graphs presented in [26]. Attack steps can also be associated to each other probabilistically, similar to the probabilistic attack graphs in [35]. With this extension the proposed dependency model allows inference of how probable attempts are and how probable attempts are to succeed, as well as inference of expected loss.

3 Probabilistic relational models

A *probabilistic relational model (PRM)* [10] specifies a template for a probability distribution over an architecture model. The template describes the metamodel for the architecture model, and the probabilistic dependencies between attributes of the architecture objects. A PRM, together with an instantiated architecture model of specific objects and relations, defines probability distributions over the attributes of the objects. The probability distributions are then used to infer the values of unknown attributes.

3.1 Architecture metamodel

An architecture metamodel M describes a set of classes, $X_1,\ldots,X_n$. Each class is associated with a set of descriptive attributes and a set of reference slots (relationships).

The set of descriptive attributes of a class X is denoted $A(X)$. Attribute A of class X is denoted $X.A$ and its domain of values is denoted $V(X.A)$. For example, in Figure 1, the class *System* has the two descriptive attributes *Availability* and *Reliability*, both with the domain *{Low, Medium, High}*.

The set of reference slots of a class X is denoted $R(X)$. We use $X.\varphi$ to denote the reference slot φ of X. Each reference slot φ is typed with the domain type $Dom[\varphi] = X_i$ and the range type $Range[\varphi] = X_j$. A reference slot φ denotes a function from X_i to X_j, and its inverse φ^{-1} denotes a function from X_j to X_i. The class *SystemAdministrator* in Figure 1 has the reference slot *Administrates* whose range is the class *System*. The reference slot $System.Administrates^{-1}$ then has range *SystemAdministrator*. Thus, the fundamental modeling constructs of PRMs are the same as in general conceptual modeling techniques.

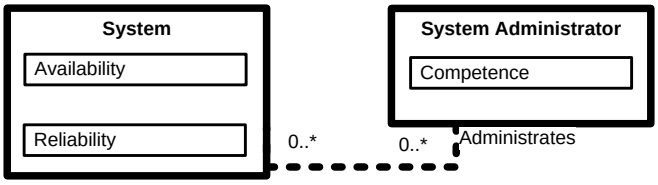

Figure 1 – An example meta-model.

3.2 Architecture instantiation

An architecture *instantiation I* (i.e. an architecture model) specifies the set of objects in each class X, and the values for attribute(s) and reference slot(s) of each object. For example, Figure 2 presents an instantiation of the meta-model described in Figure 2. It specifies a two particular *System*-object, particular *SystemAdministrator*, and the references between these. Values to the attributes are not yet ascribed.

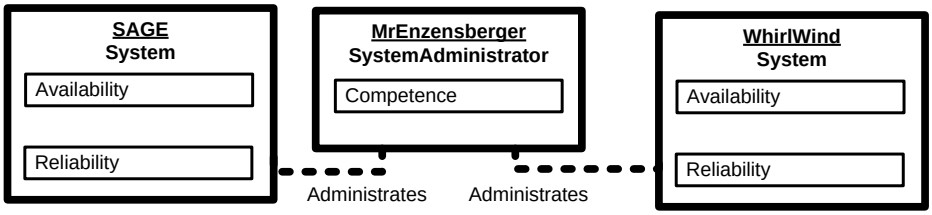

Figure 2 – An example relational skeleton, i.e. a partly instantiated model.

3.3 Probabilistic model over attributes

So far, the probabilistic relations between the attributes have not been addressed. A PRM Π specifies a probability distribution over all instantiations I of the metamodel M. This distribution is expressed in terms of a Bayesian network [45] and it consists of a qualitative dependency structure, and associated quantitative parameters.

The qualitative dependency structure is defined by associating attributes $X.A$ with a set of parents $Pa(X.A)$ that causally influence these attributes. Each parent of $X.A$ has the form $X.\tau.B$ where $B \in A(X.\tau)$ and τ is either empty, a single reference slot φ or a sequence of reference slots $\varphi_1,\ldots,\varphi_k$ such that for all i, $Range[\varphi_i] = Dom[\varphi_{(i+1)}]$. We call τ a slot chain. Note that when $X.\tau.B$ reference attributes external to the class X, it might be referencing a set of attributes rather than a single one since there may exist multiple instantiated objects of one class. In these

cases, *X.A* depends probabilistically on an aggregation function over those attributes. In general these aggregation functions could take any form. In this paper however, the logical operations *AND*, *OR*, and the arithmetic operation *SUM* (for summing).

In Figure 3 the running example is extended with two attribute dependencies. Firstly, the attribute *Availability* of the class *System* have the attribute *Reliability* of the same class as parent, meaning that the external property availability is dependent on the more internal property reliability. Secondly, *System.Availability* is also dependent on the attribute *System.Administrates*$^{-1}$*.Competence*, i.e. the competence of the system administrators that administrates the system. In the example this dependency is aggregated in terms of MAX function essentially illustrating that the system availability will be dependent on the most competent administrator. These aggregate properties are associated with a probability distribution for the case when $X.\tau.B$ is an empty set in the architecture instantiation. For instance, if *MAX(Administrates*$^{-1}$*.Competence)*could return *Low* if there is no administrator assigned.

Given a set of parents for an attribute we can now define a probability model by associating a conditional probability distribution with the attribute, *P(X.A |Pa(X.A))*. For instance, *P(System.Availability=High| Reliability=High, MAX(Administrates*$^{-1}$*.Competence)=Medium)=90%* specifies the probability that the system has a high availability given that the most competent system administrator has a medium competence and the system has a high reliability. Conditional probability distribution tables for the running example are presented in Figure 3.

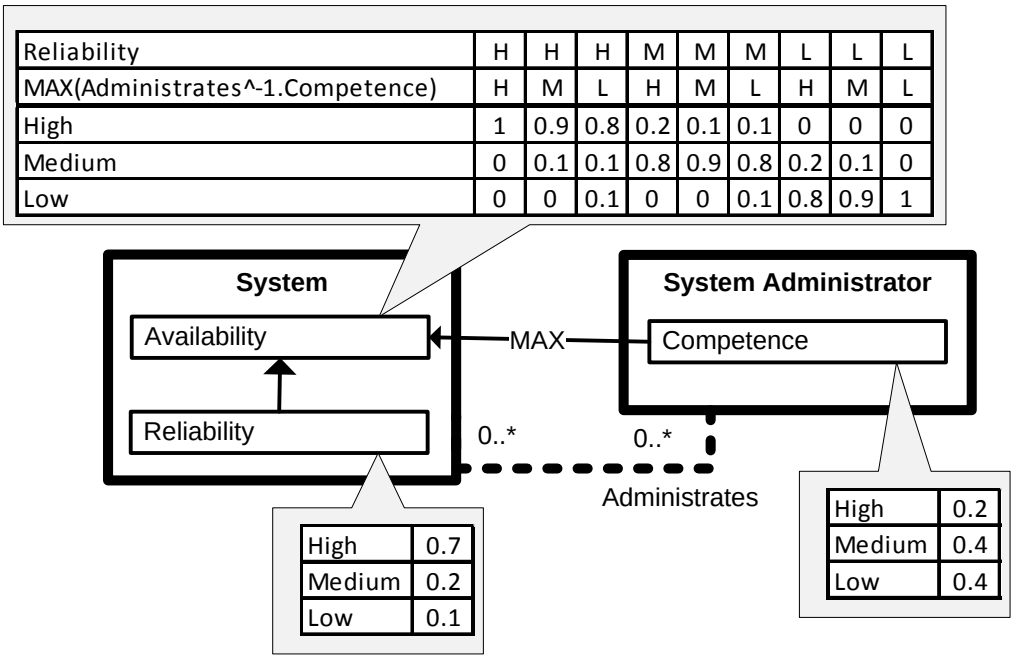

Figure 3 – An example meta-model including a qualitative and a quantitative dependency structure.

We can now define a PRM *Π* for a meta-model *M* as follows. For each class *X* and each descriptive attribute $A \in A(X)$, we have a set of parents *Pa(X.A)*, and a conditional probability distribution that represents $P_{\Pi}(X.A|Pa(X.A))$.

Given an instantiated metamodel without attribute values, σ_r, a PRM *Π* specifies a probability distribution over a set of instantiations *I* consistent with σ_r:

$$\boldsymbol{P}(\mathcal{I}|\boldsymbol{\sigma_r}, \boldsymbol{\Pi}) = \prod_{\boldsymbol{x}\in\boldsymbol{\sigma_r}(\boldsymbol{X})} \prod_{\boldsymbol{A}\in\boldsymbol{At}(\boldsymbol{x})} \boldsymbol{P}\big(\boldsymbol{x.A}\big|\boldsymbol{Pa}(\boldsymbol{x.A})\big)$$

where $\sigma_r(X)$ are the objects of each class in the instantiated metamodel. Hence, the attribute values can be inferred.

A PRM thus constitutes a formal machinery for calculating the probabilities of various architecture instantiations. This allows us to infer the probability that a certain attribute assumes a specific value, given some (possibly incomplete) evidence of the rest of the architecture instantiation.

In essence, a PRM define how a Bayesian network shall be generated over the attributes in an instance model. An extension of Bayesian networks intended to support decision-making is so called influence diagrams [47]. Influence diagrams include attributes that represents utility which are sought to be maximized or minimized. These utility nodes have a domain of utility values which are just as regular Bayesian attribute values also

ascribed probabilities. It is shown in [46] that PRMs can easily be extended to also include attributes representing the utility nodes used in influence diagrams [47]. In this paper a PRM with this extension is used.

3.4 Class inheritance and class specialization

PRMs further allow specializing classes through inheritance relationships. Classes can be related to each other using the subclass relation *«*. If *ERPSystem « System* then *ERPSystem* is a subclass of *System* and *System* is a superclass of *ERPSystem*. Let the finite set of subclasses to class *X* be *C[X]*. So if *Z, Y ∈ C[X]*, both *Z* and *Y* are subclasses of *X*. A subclass *Y* always contains the reference slots and attributes of its superclass *X*. As exemplified in Figure 4, *At(ERPSystem)* is a subset of *At(System)* and *R(ERPSystem)* is a subset of *R(System)*. The conditional probability distributions of inherited attributes can however be specialized in subclasses. A subclass can also refine the range of inherited reference slots.

For each subclass $Y \in C[X]$ and inherited attribute $B \in A(X)$, there are parents *Pa(Y.B)* and a conditional probability distribution *P(Y.B| Pa(Y.B))*. These can be equal to attributes and parents in the superclass *X*, i.e. *Pa(Y.B)=Pa(X.B)* and *P(Y.B | Pa(Y.B))= P(X.B | Pa(X.B))*, or they can be specialized to include additional parents in *Pa(Y.B)* or a different conditional probability distribution for *P(Y.B | Pa(Y.B))*. For example, as *ERPSystem ∈ C[System]*, the probability distribution for *ERPSystem.Availability* may be different (specialized) from *System.Availability*. The parents of *ERPSystem.Availability* may also be different from those of *System.Availability*. If the set of parents of an attribute is changed, i.e. $Pa(Y.B) \neq Pa(X.B)$, then the probability distribution must be specialized.

A subclass $Y \in C[X]$ may also be specialized with regard to the range of its reference slots. The reference slot is then refined. Let $X.\varphi$ be a reference slot where $Range(X.\varphi)=U$. The reference slot of the subclass, $Y.\varphi$, can have the same range as $X.\varphi$, i.e. $Range(Y.\varphi)=Range(X.\varphi)=U$. Or the reference slot of the subclass, $Y.\varphi$, can be specialized by restricting it to subclasses of *U*, i.e. $Range(Y.\varphi)=W$, where $W \in C[U]$. For instance, if *EnterpriseSystemAdministrator* is a subclass of *SystemAdministrator*, then the reference slot *EnterpriseSystemAdmin.Administrate* could be refined to the range *ERPSystem* since it is a subclass of *System*.

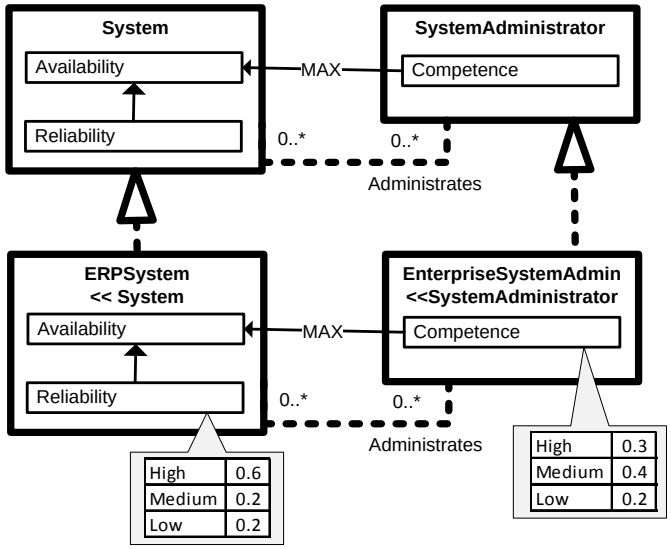

Figure 4 –Subclasses of System and SystemAdministrator. The conditional probabilitiesof ERPSystem.Reliability and EnterpriseSystemAdmin.Competence are spececialized. The reference slot EnterpriseSystemAdmin.Administrates is refined to the range ERPSystem.

4 Abstract and Concrete PRM packages

This paper presents a PRM that enables calculation of the expected loss, e.g. monetary loss, due to poor security. A set of abstract classes and an incomplete probabilistic model of how attributes of these classes depend on each other is described. This set of abstract classes will in this paper be referred to as the *AbstractPRM-package* (analogous to packages in UML) and they define a structure that is favorable to metamodels supporting security risk analysis. The AbstractPRM-package is similar to the general conceptual model in CC [21] and does for example include the classes *Asset*, *Countermeasure*, *ThreatAgent* and *Threat*. By specializing these abstract

classes into concrete subclasses a metamodel associated with a probabilistic model for security risk can be created (cf. Figure 5). In this paper the set of concrete classes is referred to as the *ConcretePRM-package*.

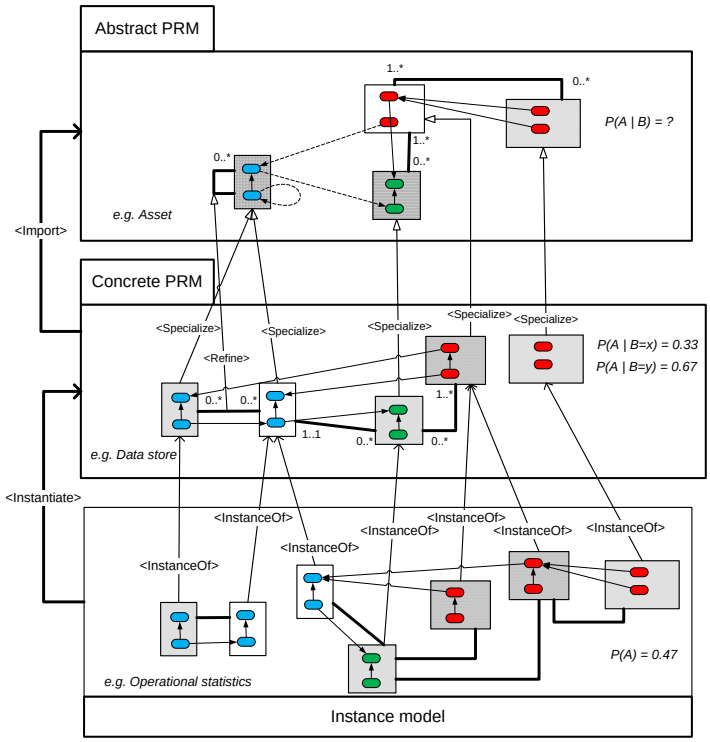

Figure 5 – The AbstractPRM-package details classes, attributes and reference slots and an incomplete probabilistic model, here represented by dashed arcs. The ConcretePRM-package specialize the classes in the AbstractPRM-package and details the probabilistic model. When the ConcretePRM is instantiated a probabilistic model over security risk can be derived.

Concrete subclasses to those in the AbstractPRM package are created using inheritance relationships. The concrete class then specialize the abstract class. A ConcretePRM-package can for instance include the countermeasure *Firewall*, the threat agent *Outsider* or the asset *DataStore*. The concrete subclasses can, as described in section 3.4, both refine the reference slots of its superclass and specialize the probabilistic model of its inherited attributes. These two features are used to create a concrete class in a ConcretePRM-package – no other operations are needed. Hence, there is no need to define new attributes of classes, and no need to identify classes that are not subclasses to these in the AbstractPRM-package. The AbstractPRM-package is associated with as set of constraints that define how the probabilistic model can be specified ConcretePRM-packages. These constraints do for instance say that the attribute *PreventiveCountermeaure.Functioning* only can be a parent of *AttackStep.PossibleToAccomplish* and that the slot chain associated with this dependency should contain reference slots that connect two assets.

If these constraints are followed, instantiations of the ConcretePRM will express probabilistic models that facilitate straightforward analysis of security risk. More specifically, the constraints ensure that if the assets and asset-to-asset relationships in a ConcretePRM are instantiated, a probabilistic dependency model can be derived. This probabilistic dependency model will express the relationships between assets, countermeasures, attack steps, threats, and threat agents in the instance model. Hence, just as abstract classes in object-oriented programming languages help a developer with blueprints and core functionality, this AbstractPRM-package helps a metamodeler to create a ConcretePRM-package for security risk analysis. It ensures that instantiations of the classes in a ConcretePRM-package produce a probabilistic dependency model that can be used to infer the probability that attacks are successful and the probability that they will be attempted. It also ensures that this can be inferred from an architectural model that only describes assets and the relationships between assets. Loss values for a successful attack can either be defined in the ConcretePRM-package, or inserted into the instance model directly. Adding such values provides the necessary means to assess expected loss.

5 An AbstractPRM-package for security risk Analysis

This chapter describes the AbstractPRM-package for security risk analysis with classes, attributes, reference slots, and attribute-dependencies. This chapter is the locus of this paper's contribution. The AbstractPRM-package consist of an architectural metamodel and a set of constraints that define how its probabilistic model over its classes may be specialized in concrete subclasses. The AbstractPRM-package is depicted in Figure 6 and described below. This description is divided in two subsections. The architectural metamodel is described first; thereafter the probabilistic model is described together with the constraints that state how subclasses can be defined in ConcretePRM-packages.

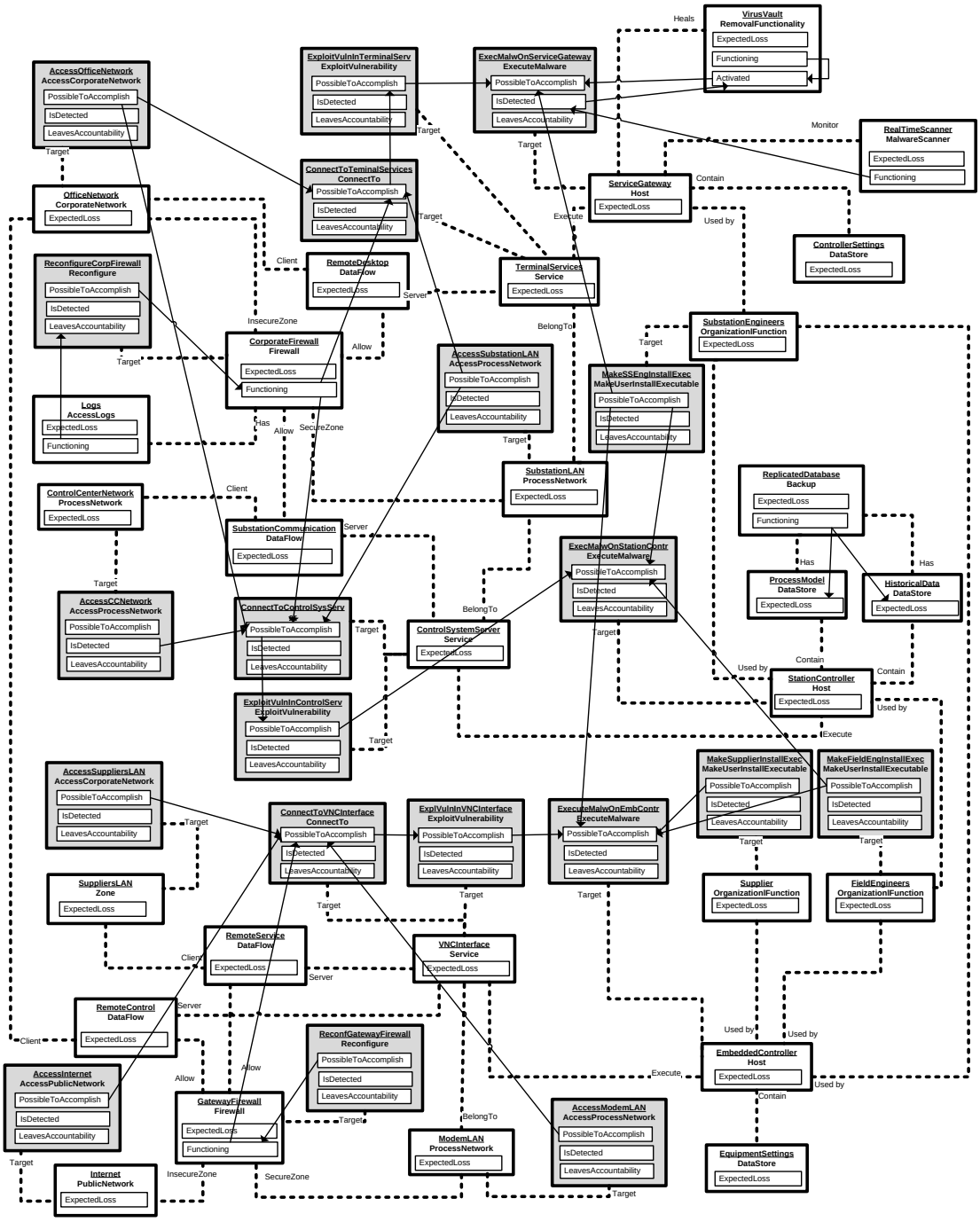

Figure 6 – An AbstractPRM-package for assessing security risk. Both the architectural metamodel and probabilistic dependencies are shown here. Some probabilistic dependencies (solid arcs) are concrete, while other (dashed arcs) are potential probabilistic dependencies that may be defined in ConcretePRM-packages. Conditional probability tables describe attribute dependencies for those attribute dependencies that are concrete in this package.

Throughout the description of the AbstractPRM-package it will be referred to concepts used in Common Criteria (CC) [21]. CC defines an internationally well established terminology for security risk assessments and its general conceptual model describes concepts that relate to security risk. In CC's general conceptual model, threat agents wish to damage and/or abuse assets. Threat agents therefore give rise to threats that are associated with an asset. Threat also increase risk to assets. Owners value assets and wish to minimize risk and they can impose countermeasures to do so. The AbstractPRM-package includes these concepts in and arranges them in a similar manner as CC's general conceptual model. However, to support quantitative risk analysis some additional constructs are included in the AbstractPRM-package: attack steps are added and related to each other, threats, assets and countermeasures; classes are given descriptive attributes; and a dependency structure is defined over these attributes. The AbstractPRM and CC's general conceptual model thus differ somewhat. These differences are just as the similarities described using CC's terminology.

5.1 Architectural metamodel

As CC's general conceptual model this metamodel relates a *Threat* to the *ThreatAgent* that gives rise to the threat. This is done with the reference slot *ThreatAgent.GiveRiseTo* with range *Threat*. Unlike CCs conceptual model however, this metamodel does not directly relate a *Threat* to an *Asset*. This is instead done through the class *AttackStep*. The AbstractPRM in Figure 6 requires a set of *AttackStep*-classes to be detailed as a part of the *Threat* using the reference slot *Threat.Includes*. This set of *AttackStep*-classes is similar to the concept "attack" that in CC's shall be used together with threat agent and asset to define a threat when the "Security Environment" is described. CC's conceptual model state that a threat agent wishes to "abuse and/or damage" assets. In the AbstractPRM-package this is represented through the reference chain *ThreatAgent.GiveRiseTo.Includes.Target*, which has the range *Asset*. Hence, each *AttackStep* is associated to the *Asset* it targets using the reference slot *AttackStep.Target*.

An *AttackStep* and *Threat* can either be possible or impossible to accomplish, hence the attributes *AttackStep.PossibleToAccomplish* and *Threat.PossibleToAccomplish* exist. A *Threat* does in addition hold the attribute *IsAttempted* and *IsRealized*. *Threat.IsAttempted* indicates if it is attempted by the threat agent or not; *Threat.IsRealized* indicates if it is realized or not. The attribute *AttackStep.IsDetected* indicates if an ongoing attack will be detected, and *AttackStep.LeavesAccountability* says if the attack step will lead to accountability, i.e. if the threat agent can be held accountable for attempting it.

CC suggests that four aspects should be used to describe threat agents. *ThreatAgent.Resources* is an aggregate of two of these – "skills" and "resources". CC also suggests that "motivation" and "opportunity" could be used in addition to these. The threat agent's motivation can in this metamodel be expressed in terms of the attribute *Threat.IsAttempted*. A motivating threat will, ceteris paribus, have a higher probability of being attempted than a threat that is not motivating. Opportunity is captured by the attribute *Threat.PossibleToAccomplish*, which express the probability that an attacker can realize the threat if this is attempted.

CC's general conceptual model includes the concept of countermeasures but does not differentiate among these with regard to how they causally reduce the risk. Outside of its conceptual model CC state that these can be seen as "Security Objectives" which are achieved by meeting "Security Functional Requirements". The AbstractPRM-package defines the class *Countermeasure* and further specialize this class to enable more detailed quantification how countermeasures depend on each other and how they influence risk. Five subclasses of *Countermeasure* are defined: *PreventiveCountermeasure* (e.g. firewall), *DetectiveCountermeasure* (e.g. intrusion detection system), *ReactiveCountermeasure* (e.g. incident handling), *ContingencyCountermeasure* (e.g. backups) and *AccountabilityCountermeasure* (e.g. logging). In relation to CC's terminology these five types of countermeasures correspond to classes of "Security Functional Requirements". How these casually reduce risk is described by the PRM's probabilistic dependency structure (cf. section 5.2).

Countermeasures can hold a value, and the class *Countermeasure* is therefore a subclass of *Asset*, i.e. *Countermeasure* « *Asset*. In addition to the inherited attribute *ExpectedLoss* which states value of the *Countermeasure*, *Countermeasure* and its subclasses also have the attribute *Functioning*. The attribute *Countermeasure.Functioning* expresses if the *Countermeasure* is working as it should – the countermeasure's correctness in CC's terminology. If a reactive countermeasure is functioning it can respond to detected events. Therefore *ReactiveCoutermeasure* also holds the attribute *ReactiveCoutermeasure.Activated* which indicates if it is put into effect or not.

Just as in CC, an *Owner* value *Asset*s through the reference slot *Owner.Value*. An *Asset* can in this model also be related to other assets using the reference slot *Asset.Association*. The risk to assets and owners is in the architecture metamodel represented by *Asset.ExpectedLoss* and *Owner.ExpectedLoss*. Both *V(Asset.ExpectedLoss)* and *V(Owner.ExpectedLoss*) are scalars that express losses, e.g. monetary losses. The

attribute *ThreatAgent.Resources* has the domain of values *{High, Medium, Low}* whose semantics is defined in ConcretePRM-packages. All other attributes in the metamodel have the domain of values *{True, False}*.

5.2 Probabilistic dependency structure and specialization constraints

As CC's conceptual model, the PRM depicted in Figure 6 contains concepts that influence security risk. This metamodel is in addition associated with a dependency structure which defines how the attributes of these concepts influence each other. To properly define a PRM it must be described how to derive the parents *Pa(X.A)* of all attributes *X.A* in an instantiated model. Attributes also need to be associated with a conditional probability distribution *P(X.A|Pa(X.A))*.

Figure 6 shows the probabilistic dependency structure as solid and dashed arcs spanning between attributes. An arc is solid when the abstract package defines how parent-attributes should be derived from an instance model, and dashed when it does not. Hence, the slot chains that define dashed arcs must be specified in packages that import the AbstractPRM-package, i.e. in ConcretePRM-packages.

Parents are left undefined when the AbstractPRM-package's metamodel does not include the information required to determine them. For example, a data backup for a data store would be a typical *ContingencyCountermeasure* in an instance model, and a data store would be a typical *Asset.* The dashed arc from *ContingencyCountermeasure.Functioning* to *Asset.ExpectedLoss* then implies that a functioning data backups influence the expected losses of the data store. This dependency structure would be reasonable if a model would instantiate an *Asset* that is a data store, this *Asset* is associated with an instance of *ContingencyCountermeasure* represent the data store's data backup. But, if the asset instead is an employee, a data backup is of little help. There is however nothing in the AbstractPRM-package that allows instance models to represent if the asset is a data store, if the countermeasure is a backup-system, or if the backup-system makes backups of the data store.

The concepts necessary to specify theories like these can however be specified in ConcretePRM packages that specialize the classes in the AbstractPRM-package. To do this a set of asset-to-asset references (*Asset.Association*) should be used. Chapter 6 shows an example of how this can be done in a ConcretePRM-package. This ConcretePRM-package does for instance say that the class *Backup* is a subclass of *ContingencyCountermeasure* and that the class *DataStore* is as a subclass of *Asset.* If these two are related with the asset-to-asset reference *DataStore.Has* it specifies that *Backup.Functioning* is a parent of *DataStore.ExpectedLoss*, i.e. that *DataStore.ExpectedLoss* has parents *DataStore.Has.Functioning*.

Asset-to-asset references are used to define the slot chains that make dashed arcs concrete (solid). For the rules dictating how the attribute-parents can be specified, recall that parents to an attribute *X.A* are defined in the form $X.\tau.B$, where $B \in A(X.\tau)$. The slot chain τ is here either empty, a single slot ρ or a sequence of slots $\rho_1,\ldots,\rho_k$ such that for all i, $Range[\rho_i]=Dom[\rho_{(i+1)}]$. Also recall that this AbstractPRM-package states that *Asset* can have zero to many references to other *Asset*s through the reference slot *Association*. Concrete specializations of the classes in the AbstractPRM-package can use refinements of the reference slot *Association* to specify concrete attribute dependencies. A refinement could for example be the reference slot *DataStore.Has*, as in the example above.

Table 1 describes the parents of attributes in the AbstractPRM. Here *a2a* is used to represent a chain of refined asset-to-asset relationships, i.e. a chain of refined *Association* reference slots. The range of the reference chain is in some cases constrained to a specific subclass of asset. If this is the case the class' name is shown within parentheses at the end of the chain. Table 1 does for example show that *Asset.ExpectedLoss* can have the parent *Asset.a2a.(ContingencyCountermeasure).Functioning*. In a ConcretePRM-package this can be used to associate a *DataStore.ExpectedLoss* with a contingency countermeasure of type *Backup*, as in the example above. The slot chain *a2a* would in this case be the single reference *Has*, thus making *DataStore.Has.Functioning* a parent of *DataStore.ExpectedLoss*. More formally, if <*AssociationRefinement*> denotes a refinement of the reference slot *Asset.Association*, then a chain of < *AssociationRefinement*> reference slots is denoted as *a2a*.

This parent structure is defined to infer a value for *Owner.ExpectedLoss*. This value will be the sum of expected losses of the *Assets* that the *Owner* values. *Asset.ExpectedLoss* is caused by *Threat*s that are realized [21], and *Threats.IsRealized* can therefore be defined as a parent of *Asset.ExpectedLoss*. As defined in [4], a threat will be realized (*Threat.IsRealized*) if it is attempted (*Threat.IsAttempted*) and if it is possible to succeed with (*Threat.PossibleToAccomplish*). For *Threat.PossibleToAccomplish* to be true, all *AttackSteps* included in the threat must be possible to accomplish. The attack steps included in a threat thus correspond to an attack path in an attack graph, and all these attack steps must be possible to accomplish if the attack path should be possible to accomplish.

Table 1 – Attributes in the AbstractPRM-package are here shown together with the slot chains defining their parents. Owner.ExpectedLoss does for example have the parent(s) Owner.Values.ExpectedLoss. A chain of refined reference slots of Asset.Association is here denoted a2a.

Attribute
Owner.ExpectedLoss
Owner.Value.ExpectedLoss
Asset.ExpectedLoss
Asset.a2a.Target^{-1}.Includes^{-1}.IsRealized
Asset.a2a.(ContingencyCountermeasure).Functioning
Threat.IsRealized
Threat.PossibleToAccomplish
Threat.IsAttempted
Threat.PossibleToAccomplish
Threat.Includes.PossibleToAccomplish
Threat.IsAttempted
Threat.Includes.LeavesAccountability
Threat.PossibleToAccomplish
AttackStep.PossibleToAccomplish
AttackStep.Target.a2a.Target^{-1}.PossibleToAccomplish
AttackStep.Target.a2a.(PreventiveCountermeasure).Functioning
AttackStep.Target.a2a.(ReactiveCountermeasure).Activated
AttackStep.Includes^{-1}.GiveRiseTo^{-1}.Resources
AttackStep.LeavesAccountability
AttackStep.a2a.(AccountabilityCountermeasure).Functioning
AttackStep.Detected
AttackStep.a2a.(DetectiveCountermeasure).Functioning
ReactiveCountermeasure.Activated
ReactiveCountermeasure.Functioning
ReactiveCountermeasure.a2a.Target^{-1}.Detected
Countermeasure.Functioning
Countermeasure.Target^{-1}.PossibleToAccomplish

*AttackStep*s can be associated to each other in a way similar to the probabilistic attack graphs described in [35]. It can be specified that accomplishment of one attack step influences the probability that another attack step will be possible to accomplish. This is specified using asset-to-asset relationships in the metamodel. The probability that the included *AttackStep*s are accomplished also depends on the *Resources* of the *ThreatAgent* [21]. The attribute *Threat.IsAttempted* is influenced by deterrent factors [26]: the personal risk associated with attempting the attack and the difficult of accomplishing with it. To represent this *Threat.IsAttempted* has parents *Threat.PossibleToAccomplish* and the OR-aggregate of *LeavesAcountability*-attributes in included *AttackStep*-objects.

As in [26,36,37], an *AttackStep* that targets a *Countermeasure* will do so to disable it. Hence, *Countermeasure.Functioning* has the parent *Countermeasure.Target*$^{-1}$*.PossibleToAccomplish*. Functioning countermeasures can lower *Owner.ExpectedLoss*, but depending on their class they will do so differently. The dashed arcs in Figure 6 show which potential parents there could be in ConcretePRM. One internal dependency exists among the subclasses of countermeasures. Time-based security prescribe that one way of preventing attacks are to detect them and trigger reactive measure to mitigate them [48]. *ReactiveCountermeasure.Activated* can therefore be influenced by *AttackStep.Detected*.

In this model the logical operations *AND*, *OR* and the arithmetic operation *SUM* are used to aggregate multiple parents into one value. The conditional probability distribution *Threat.IsAttempted* is for example specified for the OR-aggregate of all *Threat.Includes.LeavesAccountability*-parents (cf. Figure 6). The aggregate of the logical

operations *AND* and *OR* is here defined as *False* for an empty set of parents. Also let *AttackStep.PossibleToAccomplish* be *False* if *Pa(AttackStep.PossibleToAccomplish)* ∈ *At(ThreatAgent)* is the empty set.

The attributes *AssetExpectedLoss* and *Owner.ExpecredLoss* are defined as value tables with a default value of zero. All *Countermeasure* share the same conditional distribution for the attribute *Functioning*. Figure 6 describes this conditional probability, as well as all other conditional probabilities. These may, and ought to, be specialized in concrete subclasses since more accurate tables can be defined when classes are more concrete. In concrete specializations the parents of attributes targeted by dashed arcs may change as new parents are defined. In these cases the conditional probability distribution must be specialized in the subclass.

6 A ConcretePRM-package

This chapter exemplifies how a Concrete PRM can be created using the AbstractPRM-package presented in chapter 5. This is done by defining subclasses to those that exist in the AbstractPRM, by specializing the reference slots and attributes of these subclasses, and concretizing the probabilistic model using asset-to-asset references. No operations except these three are needed to create a ConcretePRM-package.

The ConcretePRM-package described in this chapter is only an illustrative example of how a Concrete-PRM package can be constructed. Hence, its structure and conditional probabilities have not been validated. The example shows a simple metamodel that can be used to analyze risks associated with malware attacks that cause loss of data. Its description is divided in four sections, each describing a part of the Concrete-PRM package. The first three sections depict different parts of the diagram in Figure 7, namely: "Firewalls, services and network zones", "Malware and anti-malware", and "Social engineering and backup data". The fourth part describes subclasses of *Threats* and *ThreatAgents* and relates these to the classes depicted in Figure 7. The fourth part is illustrated in Figure 8.

When reference slots of type *Asset.Association* inherited from the AbstractPRM-package are refined they will be given descriptive names, e.g. *DataStore.Has*. When it is necessary to distinguish between two reference slots refinements of other types of reference slots we will use an alphabetical suffix (e.g. *X.TargetA* and *Y.TargetB*).

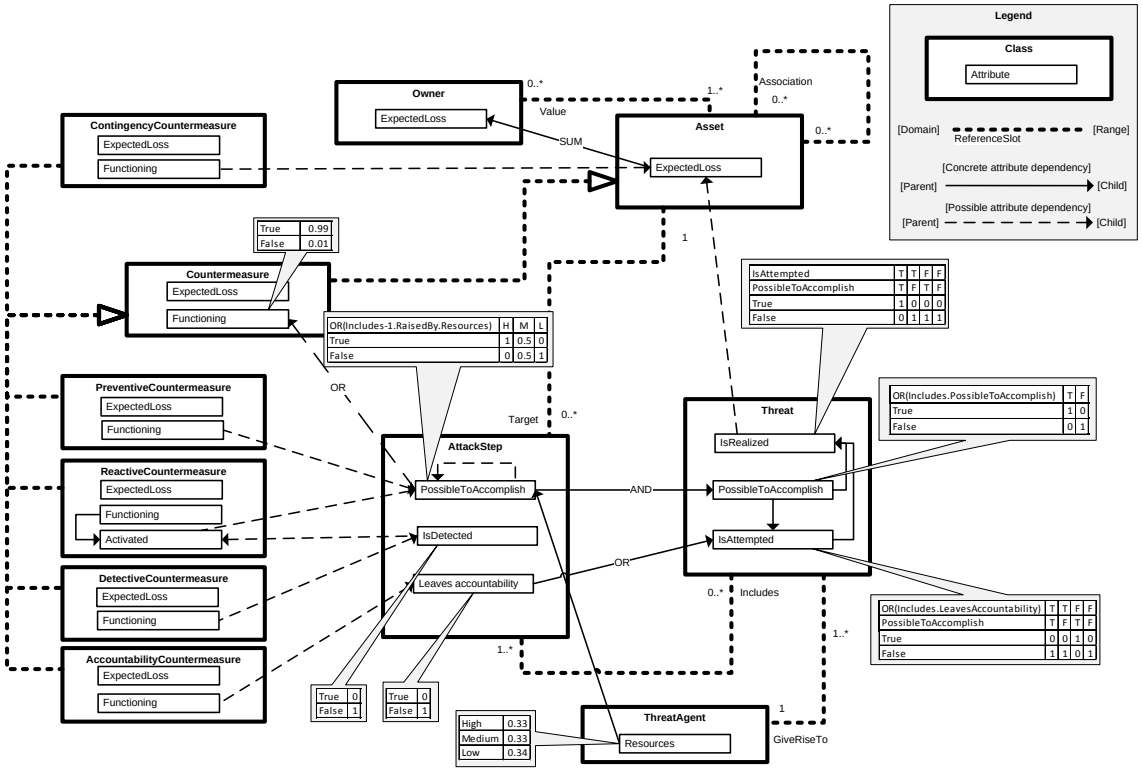

Figure 7 –These concrete classes are subclasses of those depicted in Figure 6 and concretize their attribute dependencies to create a metamodel over malware attacks that cause loss of data. The conditional probabilities for some attributes are inherited from their superclass and can be found in Figure 6. Those that are specialized are shown in this figure. Loss values (marked with question marks) are detailed in the instance model.

6.1 Firewalls, services and network zones

The primary purpose of firewalls is to control access to network addresses. They do so by blocking unwanted data flows from adjacent zones, and allow those that are wanted. With a protection scheme following the principle "deny by default", a *Firewall* will allow a number of *Data flows* to pass into the secure *Zone* from other *Zone*s. In this ConcretePRM a *Firewall* holds the reference slots *Allow, SecureZone* and *InsecureZone*, where *Range(Firewall.Allow)=DataFlow*, *Range(Firewall.SecureZone)=Zone* and *Range(Firewall.InSecureZone)=Zone*. The S*ecureZone* and *InsecureZone* here refer to the zones that are directly separated by the firewall.

A *Zone* can be targeted in the attack step *AccessZone*. Three subclasses of *Zone* are represented in the PRM: *PublicNetwork, CorporateNetwork* and *ProcessNetwork*. For these three the attack step *AccessZone* is specialized into *AccessPublicNetwork, AccessCorporateNetwork* and *AccessProcessNetwork.* This way their conditional probability distributions can be specialized.

A *DataFlow* is initiated from one *Zone* and is established to a *Service*. It has the reference slots *Client* and *Server* where *Range(DataFlow.Client)=Zone* and *Range(DataFlow.Server)* is *Zone*. A *Service* belongs to a *Zone* with the reference slot *BelongTo*, where *Range(Service.BelongTo)=Zone*.

ConnectTo is an attack step that targets a *Service*, i.e. *ConnectTo « Attack step* and *Range(ConnectTo.Target)=Service*. A *Firewall* will prevent *Services* in the secure *Zone* from unauthorized *DataFlows* where the client is in an insecure *Zone*. Hence, *Firewall « PreventiveCountermeasure* and *ConnectTo.PossibleToAccomplish* have parents *ConnectTo.TargetA.BelongTo*$^{-1}$*.SecureZone*$^{-1}$*.Functioning* and *ConnectTo.Target.BelongTo*$^{-1}$*.SecureZone*$^{-1}$*.InSecureZone.Target.PossibleToAccomplish*.

A firewall will not prevent connection attempts from the zone where the service belongs, or prevent a connection attempt that uses a data flow which is allowed by the firewall. To express this *ConnectTo.PossibleToAccomplish* has the parents: *ConnectTo.TargetA.BelongTo.BelongTo*$^{-1}$*.TargetB*$^{-1}$*.PossibleToAccomplish* and *ConnectTo.TargetA.Server*$^{-1}$*.Client.Target*$^{-1}$*.PossibleToAccomplish*.

Firewalls must function to block unauthorized data flows, i.e *Firewall.Functioning=True*. The attack step *Reconfigure* target a *Firewall* and will therefore disable them if accomplished. *Firewall* may refer to *AccessLogging* with the reference slot *Firewall.Has.* The class *AccessLogging* will influence the probability that *Reconfigure* leaves accountability. To specify this *Reconfigure.LeavesAccountability* have parents *Reconfigure.Target.Has.Functioning*.

Table 2 – Parents to attributes in the Firewall part of the ConcretePRM-package.

ConnectTo.PossibleToAccomplish
ConnectTo.TargetA.BelongTo.Target. PossibleToAccomplish ConnectTo.TargetA.Server^{-1}.Client.TargetB^{-1}.PossibleToAccomplish ConnectTo.TargetA.BelongTo.SecureZone^{-1}.InsecureZone.Target.PossibleToAccomplish ConnectTo.TargetA.BelongTo. SecureZone^{-1}.Functioning
Firewall.Functioning
Firewall.Target^{-1}.PossibleToAccomplish
Reconfigure.LeavesAccountability
Reconfigure.Target.Has.Functioning

6.2 Malware and anti-malware

Services can be exploited to execute malware on the hosts that run them. *ExploitVulnerability* has the reference slot *TargetB*, where *Range(ExploitVulnerability.TargetB)=Service*, and *Host* have the reference slot *Host.Run* where *Range(Host.Run)=Service*. The reference slots *ExecuteMalware.Target* and *MalwareScanner.Monitor* have the range *Host*.

A service's vulnerability can only be exploited if a connection can be established to it. Hence, *ExecuteMalware.Target.Run.TargetB*$^{-1}$*.PossibleToAccomplish* is a parent of *ExecuteMalware.PossibleToAccomplish.* The attack step *ExecuteMalware* can be detected if the host is monitored with a *MalwareScanner*. If detected, *ExecuteMalware.PossibleToAccomplish* will be influenced by removal functionality on the host *RemovalFunctionality.Activated*. The rules for these dependency relationships are shown in Table 3.

Table 3 – Parents to attributes in the Anti-malware part of the PRM.

ExploitVulnerability.PossibleToAccomplish
ExploitVulnerability.TargetB.TargetA^{-1}.PossibleToAccomplish
ExecuteMalware.PossibleToAccomplish
ExecuteMalware.Target.Run.TargetB^{-1}.PossibleToAccomplish
ExecuteMalware.Target.Heal^{-1}.Activated
ExecuteMalware.Detected
ExecuteMalware.IsTargetIn^{-1}.Monitor^{-1}.Functioning
RemovalFunctionality.Activated
RemovalFunctionality.Functioning
RemovalFunctionality.Heals.Target^{-1}.Detected

6.3 Social engineering and backup data

In addition to technical concepts, such as firewalls and network addresses, the AbstractPRM-package allows organizational and human elements of security to be included in ConcretePRM-packages. In this example (cf. Figure 7), the class *OrganizationalFunction* represents a role within the organization. *OrganizationalFunction* has the reference slots *Use* and *CoveredBy*, where *Range(Use)=Host* and *Range(CoveredBy)=AwarenessProgram*.

An *OrganizationalFunction* can be targeted by the attack step *MakeUserInstallExecutable*, so *Range(MakeUserInstallExecutable.Target)= OrganisationalFunction*. The attribute *ExecuteMalware.PossibleToAccomplish* is dependent on whether it is possible to make the targeted host's users install an executable in a host. This in turn is influenced by whether this person is covered by a security awareness program.

Host has the reference slot *Contain* with range *DataStore*. The class *DataStore* has the reference slot *Has*, where *Range(DataStore.Has)=Backup*. The expected loss of a data store is influenced by if its backups are functioning. This value can for example represent the monetary loss associated with the conditions in parent-attributes. If general loss values can be identified, and these are expected to be as accurate as those that can be provided by the person applying the ConcretePRM-package, the loss values associated with successful attacks can be specified in the ConcretePRM-package. In this ConcretePRM-package the class *DataStore* is however vaguely defined. The monetary loss associated with the destruction of different *DataStore*-objects is this example collected together with the instance model instead. The question marks in the definition of this *DataStore.ExpectedLoss* mark that this information should be provided when the architecture model is instantiated.

Table 4 details the rules to determine parents of this attribute as well as other attributes in this part of the ConcretePRM-package. The parent *DataStore.Contain^{-1}.Target^{-1}.Includes^{-1}.IsRealized* refers to the *Threat* which causes the loss and is explained in more detail in section 6.4.

Table 4 – Parents to attributes in the social engineering part of the PRM.

DataStore.ExpectedLoss
DataStore.Has.Functioning
DataStore.Contain^{-1}.Target^{-1}.Includes^{-1}.IsRealized
ExecuteMalware.PossibleToAccomplish
ExecuteMalware.IsTargetIn^{-1}.UsedBy.IsTargetIn.PossibleToAccomplish
MakeUserInstallExecutable.PossibleToAccomplish
MakeUserInstallExecutable.IsTargetIn^{-1}.CoveredBy.Functioning
DataStore.Contain^{-1}.Target^{-1}.Includes^{-1}.IsRealized

6.4 Threats and Threat Agent

The AbstractPRM-package allows a ConcretePRM-package to specialize *ThreatAgent* and *Threat* into subclasses. *ThreatAgent* can be specialized to define the probability distribution of the attribute *Resources*, and to define the semantics of its states. The class *Threat* can be specialized to: define *Threat.IsRealized* as a parent to some *Asset.ExpectedLoss*, to refine the reference slot $Threat.GiveRiseTo^{-1}$, to refine the reference slot *Threat.Includes*, and to specialize the conditional probability of *Threat.IsAttempted*.

Subclasses of *Threat* and *ThreatAgent* and how these relate to other classes in the ConcretePRM package is shown in Figure 8. The conditional probabilities for the *IsAttempted*-attributes here represent the probability that an attack will be attempted over the duration of one year.

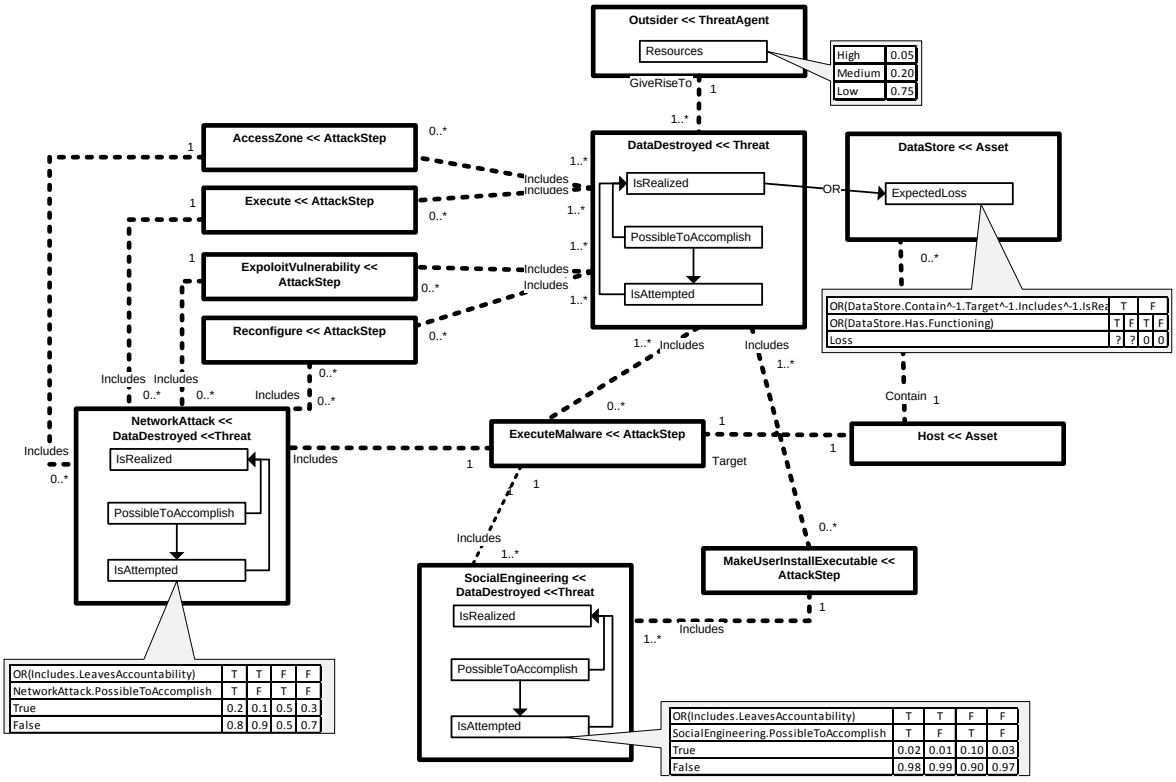

Figure 8 – Threats and ThreatAgent in the ConcretePRM-package. The refined range of the Includes-slot in SocialEngineering and NetworkAttack is here defined together with their multiplicities. The figure also shows specialized probability distributions and that DataDestoryed.IsRealized is a parent to DataStore.ExpectedLoss. Attributes of AttackStep-specializations and associated attribute dependencies is not shown in this figure.

DataDestoroyed is a subclass of *Threat*. The attribute *DataStore.ExpectedLoss* has parents $DataStore.Store^{-1}.Target^{-1}.Include^{-1}.IsRealized$, i.e. if this threat is realized *DataStore.ExpectedLoss* is influenced. We also let Outsider be a subclass of *ThreatAgent* and *Outsider.GiveRiseTo* be refined so that *Range(Outsider.GiveRiseTo)= DataDestroyed*. The probability distribution for *Outsider.Resources* is shown in Figure 8 and its states are defined as: (*High*) a professional cyber-criminal spending up to one week, (*Medium*) a professional cyber-criminal spending up to one hour, (*Low*) a beginner spending one hour.

To refine the *AttackStep*s a *Threat* might include we let *NetworkAttack* and *SocialEngineering* be two subclasses of *DataDestroyed*, i.e. *NetworkAttack « DataDestroyed* and *SocialEngineering « DataDestroyed*. Figure 8 specifies the conditional probabilities for *IsAttempted* for these two threats. We let the range of *SocialEngineering.Includes* and *NetworkAttack.Includes* be defined as in Figure 8. These refinements do for instance say that *NetworkAttack.Includes* may refer to any number of *Reconfigure*-instances, but not to an instance of *MakeUserInstallExecutable*.

7 Instance model and security risk analysis

The AbstractPRM-package provides a basis for creating metamodels that supports probabilistic inference of security risk. This chapter exemplifies how inference is supported by applying the ConcretePRM-package described in chapter 6 to a case study.

The case study and the resulting instance model are described in section 7.1. In section 7.2 it is shown how probabilistic reachability analysis can be performed on the instance model. This method of analysis is similar to the reachability analysis performed on attack graphs and produce probabilities on the possibility to accomplish different attack steps. The AbstractPRM-package does in addition to this also allow inference of expected loss. Section 7.3 describe how expected loss can be inferred from instance models that instantiate ConcretePRM-packages.

7.1 Case study

The case study was carried out at an electric power utility in Sweden during November 2009. The scope of this study was one of the critical substations within the utility. This substation was modeled with ConcretePRM-package described in chapter 6. To create the instance model, interviews with system administrators were conducted and system documentation was investigated.

To create the instance model subclasses of *Asset* and reference slots where both range and target is an *Assets* (i.e. refinements of *Asset.Association*) must be specified. This, together with loss values of different *DataStore*-objects, was the only information collected in the case study. In Figure 9 white boxes represent the objects that instantiate a subclass of *Asset*. The dashed lines between these white boxes are the instantiated reference slots.

Four relevant *Zones* are located outside of the substation. The *OfficeNetwork* is the insecure side of the *CorporateFirewall*. The *CorporateFirewall* allow the data flow *RemoteDesktop* to pass through from the *OfficeNetwork* to the service *TerminalServices*, and the data flow *SubstationCommunication* from the zone *ControlCenterNetwork* to reach *ControlSystemServer*. The *GatewayFirewall* is connected to the Internet and allow data to pass from both the *OfficeNetwork* and the *SuppliersLAN*.

Within the substation there are two process networks: the *SubstationLAN* and the *ModemLAN*. The services *ControlSystemServer* and *TerminalServices* belong to the *SubstationLAN*. The *ControlSystemServer* is executed by the host *StationController* and is the server for *SubstationCommunication*. The *StationController* contain a *ProcessModel*. This data store has a backup – the *ReplicatedDatabase*. The service *TerminalServices* is executed by the host *ServiceGateway* and is the server for the data flow *RemoteDesktop*. The *ServiceGateway* contains *ControllerSettings* and is used by *SubstationEngineers* to reconfigure information technology within the substation. The *ServiceGateway* is monitored by a *RealTimeScanner* and a *VirusVault* can heal it if malware is found on it. The service *VNCInterface* belongs to the ModemLAN and is executed by the host *EmbeddedController*. This *EmbeddedController* contains another data store – *EquipmentSettings*. In a *BackupCabinet* there are backups of both *EquipmentSettings* and *ControllerSettings.*

Three organizational functions use hosts within the substation: *Supplier*, *FieldEngineers* and *SubstationEngineers*. The *EmbeddedController* is used by Supplier and *FieldEngineers*; the *ServiceGateway* is used by *SubstationEngineers*; the *StationController* is used by *SubstationEngineers* and *FieldEngineers*. None of these are covered by an awareness program.

The tables in Figure 9 present the loss (in Swedish krona, SEK) that would be experienced if the data stores would be destroyed. These state the loss under the conditions that a backup exist, and that it does not exist. As can be seen from Figure 9 the impact of backups on the expected loss is substantial for the attributes *ControllerSettings.ExpectedLoss* and *ProcessModel.ExpectedLoss*. The impact from a backup on the attribute *EquipmentSettings.ExcepectLoss* is less, both in relative and absolute terms. This is due to the complicated nature of these settings and the manual labor required to restore them from a backup.

The *AttackSteps* relevant for this architecture model can be derived and instantiated based on the *Asset*-objects and the reference slots between *Asset*-objects. The multiplicities associated with reference slots constrain the instance model, and provide support for deriving the attack steps that should be included in an instantiation. As suggested by the AbstractPRM, the *AttackStep*-subclasses in the ConcretePRM-package refer to exactly one *Asset* with the reference slot *Target*. This makes it possible to identify and instantiate the *AttackStep*-objects that an instantiated *Asset*-subclass should be associated to – for each *Asset*-object in the instance model, an object of each class in $Range(Asset.Target^{-1})$ should be added and referred. Hence, based on instantiations of *Assets* and asset-references, associated *AttackStep* instances can be derived. The attack steps associated with this architecture models are colored grey in Figure 9.

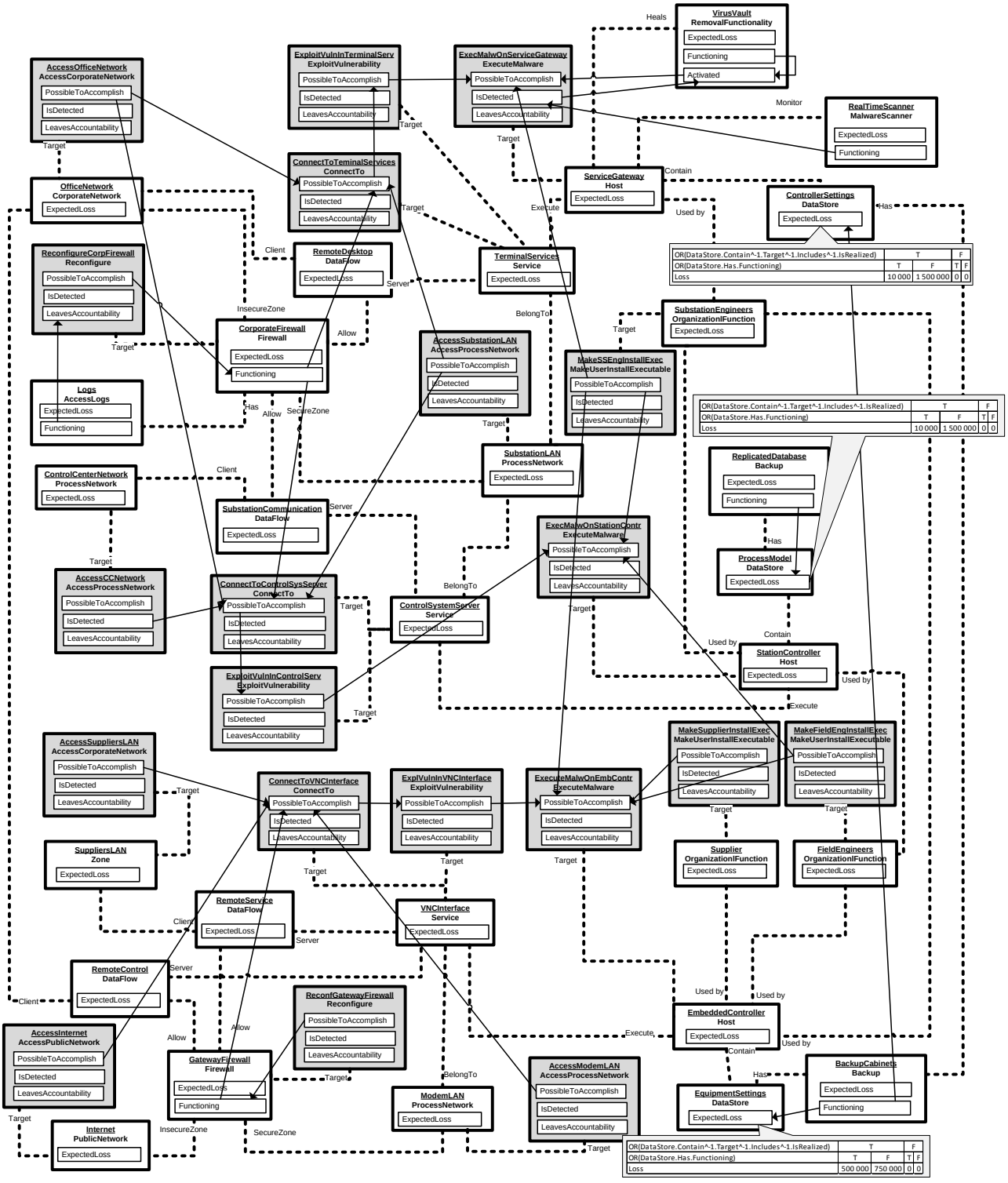

Figure 9 – The case study represented through the ConcretePRM depicted in Figure 7. AttackStep-instances (toned gray) can be derived from the ConcretePRM-package together with their references.

With this instance model as a basis, different architectural changes can be assessed in terms of their impact on reachability and expected loss. The ConcretePRM-package used in this case study specifies that only organizational functions that use a host can be made to install executables on them. It also specifies that if it is more difficult to make users install executables if their organizational functions are covered by awareness programs. Two architectural changes were assessed in this case study: 1) the impact of making sure that substation engineers do not use the embedded controller and 2) covering substation engineers with an awareness program. Objects instantiating *Owner*, *Threat* and *ThreatAgent* are not shown in Figure 9. These are instead introduced in section 7.2 and 7.3, together with the analysis.

7.2 Probabilistic reachability analysis

The dependencies between objects of class *Countermeasure* and *AttackStep* is given by an instance model. These relationships make it possible to create a probabilistic graph for reachability analysis where the possibility. This reachability analysis assesses the possibility to accomplish different attack steps targeting assets in the instance model.

If *I* is the architecture instantiation of a ConcretePRM, let I^{RA} include all objects in *I* that are of class: *AttackStep*, *PreventiveCounteremeasure*, *DetectiveCountermeasure* or *ReactiveCountermeasure*, including their attributes and internal attribute dependencies. Also let the objects *T* and *TA* be included in I^{RA}, where *T* is an instance of a subclass to *Threat* ($T \in C[Threat]$) and *TA* be an instance of a subclass to *ThreatAgent* ($TA \in C[ThreatAgent]$). Let *T.Includes* point to all *AttackStep*-objects in its range, and let *TA.GiveRiseTo* point to *T*.

This yields a graph, I^{RA}, which can be used to infer the probabilities of attributes *PossibleToAccomplish* and *Detected* in all instances of *AttackStep*. This inference is performed under the assumption that all attack steps are attempted.

Figure 10 depicts the graph I^{RA}, excluding *TA* and *T*, for the instance model *I* in Figure 9. Let *T* be of class *DataDestroyed*, and *TA* be of class *Outsider*. An instance of *DataDestroyed* (*InstanceOfTA*) then refers to all *AttackStep*-objects in Figure 9 with the reference slot *Includes*, and an instance of *Outsider* (*InstanceOfT*) refers to the instance of *DataDestroyed*.

The probabilistic model associated with I^{RA} can for example be used to infer if attack steps are possible to accomplish given different *TA.Resources*, or to infer which attack steps that can be accomplished when the attributes of attack steps and countermeasures are set to specific values, i.e. evidence has been provided on the state of these attributes. In Figure 10 the reachability analysis has been performed under the assumption that threat agent has medium resources, i.e. *TA.Resources=Medium*. Figure 10 does for instance show that *P(ExecuteMalwOnEmbContr.PossibleToAccomplish)=50 %* under these conditions. It also shows that the possibility to exploit vulnerabilities in the services is significantly smaller than the possibility to make users install executables on hosts.

Two architectural modifications were assessed in this case study. The first architectural modification was to make sure that substation engineers do not use the embedded controller. This would according to the ConcretePRM-package reduce *P(ExecuteMalwOnEmbContr.PossibleToAccomplish)* from 50 % to 37 %. The second modification, covering substation engineers in an awareness program, would reduce *P(MakeSSEngInstallExec.PossibleToAccomplish)* from 20 % to 10%. This would in turn reduce *P(ExecuteMalwOnServiceGateway.PossibleToAccomplish)* to 10 %, *P(ExecMalwOnStationContr.PossibleToAccomplish)* to 28 % and *P(ExecuteMalwOnEmbContr.PossibleToAccomplish)* to 44 % . The impact of architectural changes like these on reachability is assessed by adding or removing assets and asset-to asset relationships in the instance model.

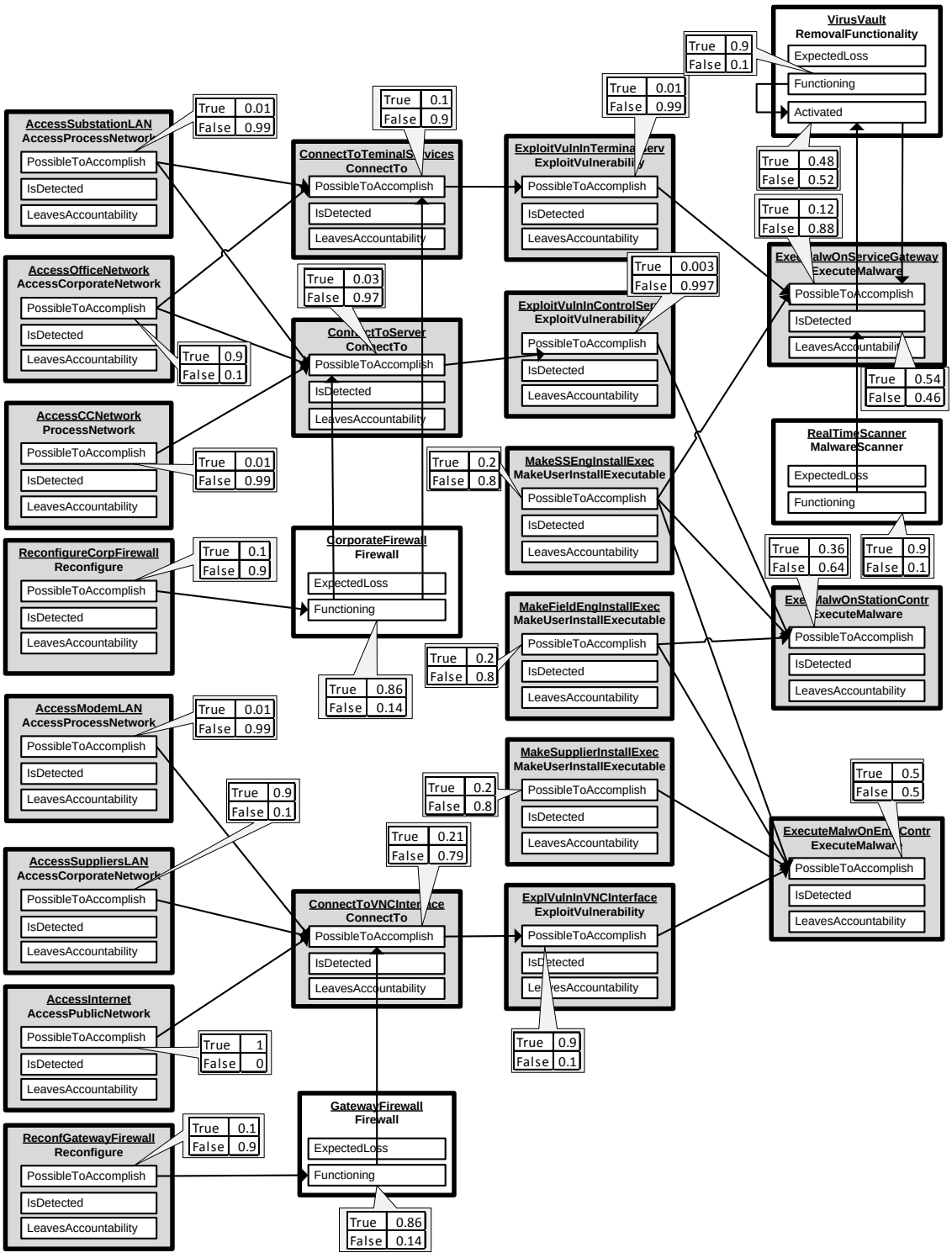

Figure 10 – Reachability graph comprising of attack steps, preventive countermeasures, detective countermeasures and reactive countermeasures along with their attribute-relationships. Here Adversary.Resources (not shown in the figure) is set to medium.

7.3 Loss expectancy

The method described in section 7.2 allows inference of the probability that attack steps are possible to accomplish under the assumption that the threat agent attempts to perform all attack steps. Although the probabilities that are inferred with this analysis method provide an indication of security, just as reachability analysis performed on attack graphs and attack trees, it does not capture security risk (*Owner.ExpectedLoss*). If

risk is the variable sought, it should also be assessed if the threats will be attempted (*Threat.IsAttempted*) and threats should be related to the losses they cause when realized. Countermeasures that deter threat agents from attempting attacks (*AccountabilityCountermeasure*) and countermeasures that limit the loss (*ContingencyCountermeasure*) are also of relevance when this is assessed. By instantiating the threats one wish to include in the analysis these factors can be included in the analysis. The expected loss from different threats can be assessed. In this case study we instantiated objects the classes of *NetworkAttack* and *SocialEngineering*.

In Figure 11 two *Threat*-instances are shown. In the first *Threat1* includes *MakeSSEngInstallExec* and *ExecMalwOnServiceGateway*; in the second *Threat2* includes *MakeSSEngInstallExec* and *ExecMalwOnStationContr*. For each plausible instance like these, the probability *P(SocialEngineering.IsRealized)* can be inferred and the value for the expected losses associated with different assets can be calculated. The probabilistic dependency structure infers the expected losses 619 SEK and 1385 SEK for these two threats.

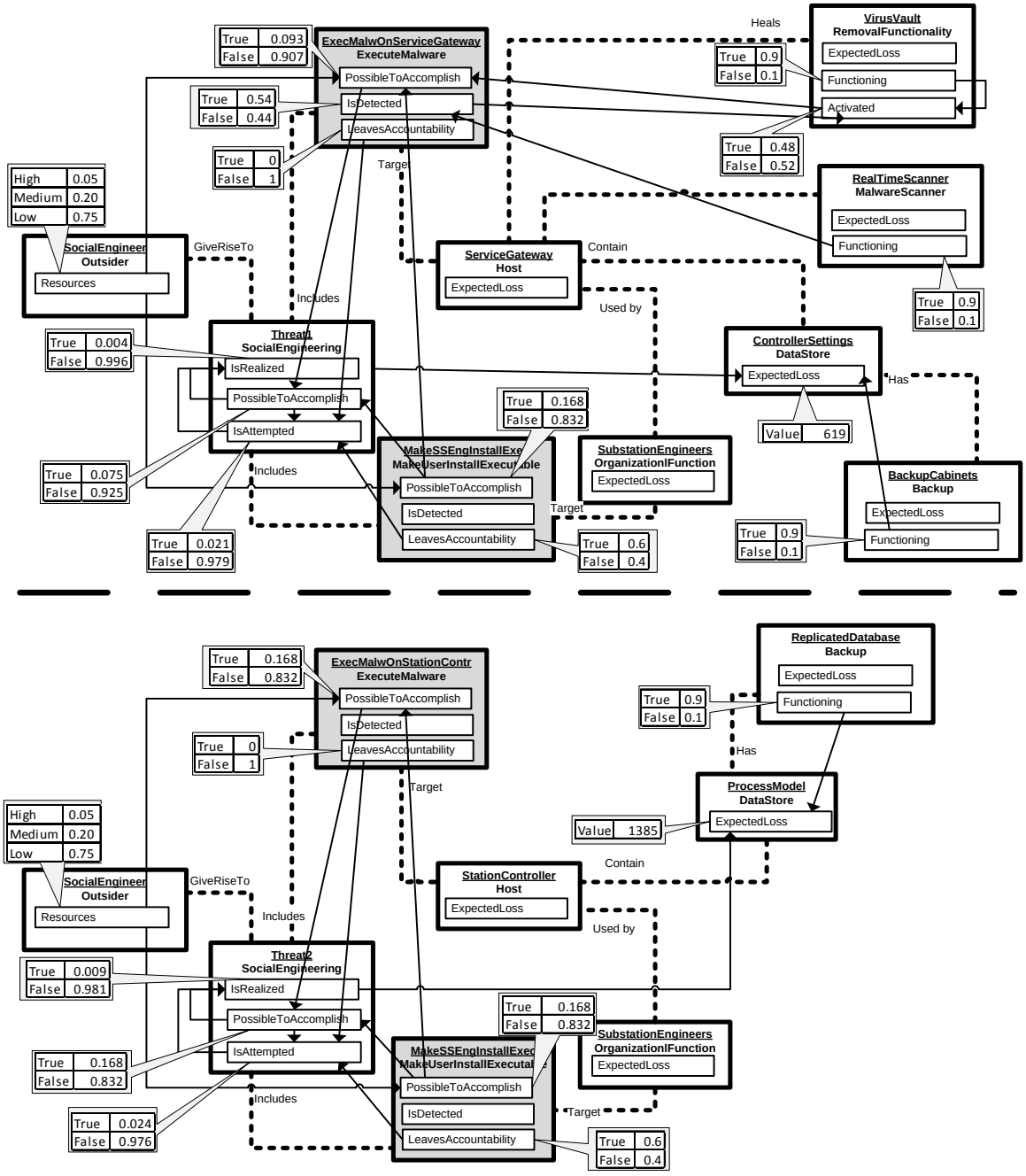

Figure 11 – Example of instantiated SocialEngineering-threats that corresponds to Threat1 and Threat2. Only those objects that are of relevance to these threats are included in the figure.

In this case study 17 different threats were found relevant for the substation in question. Six are of the class *SocialEngineering* and eleven of the class *NetworkAttack*. Table 5 and Table 6 list these together with the

expected losses associated with them. Creating the object *ElectricUtility* of class *Owner* and referring this to all *DataStore*-instances aggregates these losses to the attribute *ElectricUtility.ExpectedLoss*.

Table 5 – Instantiations of SocialEngineering and associated expected loss. An X denotes that the attack step is included in the threat.

AttackStep\Threat	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
MakeSSEngInstallExec	X	X	X			
MakeSupplierInstallExec				X		
MakeFieldEngInstallExec					X	X
ExecMalwOnServiceGateway	X					
ExecuteMalwOnStationContr		X			X	
ExecuteMalwareOnEmbContr			X	X		X
Expected loss ($)	**620**	**1385**	**4573**	**689**	**1385**	**4573**

Table 6 – Possible references and assessed instantiations of NetworkAttack. An X denotes that the attack step is included in the threat.

AttackStep\Threat	**T7**	**T8**	**T9**	**T10**	**T11**	**T12**	**T13**	**T14**	**T15**	**T16**	**T17**
AccessOfficeNetwork	X	X		X	X						
AccessCCNetwork						X	X				
AccessSubstationLAN			X					X			
AccessSuppliersLAN									X		
AccessInternet										X	X
AccessModemLAN											
ReconfigureCorpFirewall		X			X		X				
ReconfGatewayFirewall											X
ConnectToTerminalServices	X	X	X								
ConnectToControlSystemServer				X	X	X	X	X			
ConnectToVNCInterface									X	X	X
ExploitVulnInTerminalServ	X	X	X								
ExploitVulnInControlServ				X	X	X	X	X			
ExploitVulnVNCInterface									X	X	X
ExecMalwOnServiceGateway	X	X	X								
ExecuteMalwOnStationContr				X	X	X	X	X			
ExecuteMalwareOnEmbContr									X	X	X
Expected loss ($)	**379**	**243**	**57**	**71**	**75**	**94**	**60**	**790**	**1851**	**279**	**810**

If the architecture is changed so that substation engineers do not use the embedded controller this influences the third threat (T3). T3 would in this case be associated with an expected loss of zero. Covering substation engineers in an awareness program would influence threat one (T1), two (T2) and three (T3). The expected losses from these threats would be reduced to 390 SEK, 689 SEK and 2275 SEK respectively. With these predictions as a basis, the first architectural change lowers expected losses by 4573 SEK and the second alternative would reduce losses by SEK 3224.

8 Discussion

This paper proposes the use of PRMs to perform security risk analysis based on architecture models and present a package of abstract base classes for PRMs with this purpose. This chapter will discuss the expressiveness of the proposed class-structure and some practical issues associated with the suggested approach.

The AbstractPRM-package presented herein prescribes the classes, attributes, class-references, and attribute dependencies a ConcretePRM-package should include. Although the AbstractPRM-package thereby constrains the structure of a ConcretePRM-package, it does not dictate what level of detail a ConcretePRM-package should have. A ConcretePRM-package could include attack steps on abstraction levels such as “Run code”, “Exploit buffer overflow vulnerability to run code” or “Exploit VU#238019 to run arbitrary code”. Similarly, a ConcretePRM-package can specify assets, countermeasures, threats and threat agents on any level of abstraction. This flexibility is a result of the possibility to represent uncertain variable-relationships probabilistically. A more detailed model would typically enable more informative conditional probability distributions. Given that the conditional probabilities are accurate this would make a ConcretePRM-package able to produce more informative predictions when it is instantiated. For example, a detailed ConcretePRM-package might be able to predict the possibility to accomplish with an attack to 96% or 1 % for two architectures. A less detailed ConcretePRM-package might be able to predict this probability to 70% or 35 %.

Although detailed ConcretePRM-package would allow more predictive power, more effort is required both when the ConcretePRM-package is created and when the Concrete PRM-package is instantiated. Hence, informative predictions must be balanced against cost of identifying accurate conditional probabilities for it and the practitioners’ cost when creating instance models with it. Another trade-off is the utility and relevance of the PRM’s predictions to practitioners. A PRM that only covers a limited scope might be able to offer more informative predictions than one with a wider scope and less detail. However, a limited scope increases the risk of sub optimizing the architecture design decisions from an enterprise-wide perspective.

One feature of the proposed AbstractPRM-package is that it includes inference of the probability that attacks are attempted. The threat agent’s decision model is here compactly represented by a probability distribution which states if the threat will be attempted given the probability that the attack succeeds and that it leads to accountability. Properties of threat agents are also compactly represented in the AbstractPRM. Clearly, the representation of threat agents’ resources on the scale High/Medium/Low cannot capture all distinguishing characteristics of threat agents. This attribute is however, just as threat agents’ decision model, a research field on its own. This AbstractPRM-package does not elaborate on these two fields, but instead provides a clear-cut interface to them.

A software tool [49] has been developed to support the creation and instantiation of PRMs based on the inference engine SMILE [53]. Instantiated PRMs can become quite large, but there exist today methods for solving also very large Bayesian networks [54]. Future extensions of this tool include support to automatically instantiate relevant attack steps and threats for a particular instance model. To be of practical use for decision makers though, ConcretePRM-packages must be specified. The dependencies among variables in the security field can be obtained from sources such domain experts, literature, experiments, vulnerability statistics, security exercises (e.g. red team-blue team exercises), or a combination of sources like these. Models can also be updated as new threats or countermeasures emerge.

To create a ConcretePRM-package where the qualitative structure is optimal with regard to security risk prediction and the conditional probabilities represent an undisputable truth is of course extremely difficult, if not impossible. If domain experts judgment has been used to define conditional probabilities its output will also be of subjective nature. However, to be of practical use it is sufficient if a ConcretePRM-package captures the knowledge that is available in the security field and thereby provides the decision maker with a tool that improves security risk analysis activities. Also, decision makers are often interested in how different architectural scenarios are ordered when it comes to security, for example if the to-be architecture is better or worse than the as-is architecture. In that case the exact values of the predictions are not their most important quality, but rather that scenarios are correctly ordered with respect to their security risk.

The mere possibility to express and quantify how security theories relate to different architectures is another feature offered by the AbstractPRM-package. Security theories from diverse fields can be expressed in ConcretePRM-packages, and these theories can be consolidated with each other by integrating their packages. Furthermore, since PRMs in their pure form are versatile, ConcretePRM-packages can also be integrated with PRMs that express theories from other fields. For example theories on how costs, business value or modifiability relates to different architectures. This would allow decision makers to make informed decisions regarding the security risk associated with different enterprise architectures; while at the same time take other concerns into consideration.

9 Conclusions

PRMs allow architectural metamodels to be coupled to a probabilistic inference engine. This makes it possible to specify how the state of object’s attributes depends on the state of other attributes in an architectural model. This

paper proposes a package of abstract PRM-classes that specify how probabilistic models should be coupled to architectural metamodels to enable security risk analysis.

The versatility provided by the probabilistic side of PRMs makes it possible to specify security theories on any abstraction level and to couple these with an architectural metamodel with the proposed set of abstract classes. Concrete classes can be developed by creating subclasses to the package of abstract classes and under a set of constraints associate these with a probabilistic model. By specializing the abstract classes into concrete classes an architectural metamodel is defined and associated with formal machinery for assessing security risk from its instantiations. The structure inherited from the abstract classes also ensures that this formal machinery can calculate security risk from an instance model that only specifies assets and asset-relationships specified in an architectural model. Hence, the person instantiating the instance model is not required to quantify security attributes or provide information on vulnerabilities for security risk to be assessed.

References

[1] J. J. C. H. Ryan, D. J. Ryan, Expected benefits of information security investments, Computers & Security, 25 (2006) 579-588.

[2] T. Tsiakis, G. Stephanides, The economic approach of information security, Computers & Security, 24 (2005) 105-108.

[3] H. Cavusoglu, B. Mishra, and S. Raghunathan, A model for evaluating it security investments, Communications of the ACM, 47 (2004) 87–92.

[4] L. A. Gordon, M. P. Loeb, Managing Cybersecurity Resources: A Cost-Benefit Analysis, Mcgraw-Hill, New York, USA, 2006.

[5] W. Ozier, Risk analysis and assessment, in Information security management handbook, 4th ed, Auerbach, Boca Raton, USA, 1999, pp. 247–285.

[6] W. Huaqiang, F. Deb, C. Olivia, R. Chris, Cost benefit analysis for network intrusion detection systems, CSI 28th annual computer security conference, Washington, DC, 2001.

[7] C. Iheagwara, The effect of intrusion detection management methods on the return on investment, Computers and Security, 23(2004) 213–228.

[8] I. Hogganvik, A Graphical Approach to Security Risk Analysis. Oslo, Norway, Norway: University of Oslo - Faculty of Mathematics and Natural Sciences, 2007.

[9] O. Sheyner, Scenario Graphs and Attack Graphs. PhD Thesis, Carnegie Mellon University, 2004.

[10] N. Friedman, L. Getoor, D. Koller, A. Pfeffer, Learning probabilistic relational models, in In International Joint Conferences on Artificial Intelligence, pp. 1300-1309, 1999.

[11] Object Management Group (OMG), Unified Modeling Language (UML), 2009.

[12] Object Management Group (OMG), OMG Systems Modeling Language (OMG SysML), 2008.

[13] Object Management Group (OMG), Business Process Modeling Notation, 2009.

[14] G. Sindre, A. L. Opdahl, Eliciting security requirements with misuse cases, Requirements Engineering, 10 (2005) 34–44.

[15] J. McDermott and C. Fox, Using abuse case models for security requirements analysis, in Proceedings of the 15th annual computer security applications conference, pp. 55, Phoenix, Arizona, USA, 1999.

[16] J. McDermot, Abuse-case-based assurance arguments, in Proceedings of the 17th annual computer security applications conference, pp.0366, New Orleans, Los Angeles, USA, 2001.

[17] T. Lodderstedt, D. Basin, J. Doser, SecureUML: A UML-Based Modeling Language for Model-Driven Security, in «UML» 2002 - The Unified Modeling Language. Springer Berlin / Heidelberg, 2002.

[18] K. M. Trevisani, R. E. Garcia, SPML: A Visual Approach for Modeling Firewall Configurations, in Modeling Security Workshop, Toulouse, France, 2008. Avaliable at: http://www.comp.lancs.ac.uk/modsec/papers/modsec08_submission_21.pdf

[19] H. Mouratidis, P. Giorgini, G. Manson, I. Philp, A Natural Extension of Tropos Methodology for Modelling Security, in Proceedings of the Agent Oriented Methodologies Workshop, Seattle, USA, 2002.

[20] J. Jürjens, Secure Systems Development with UML. Berlin Heidelberg: Springer-Verlag, 2005.

[21] ISO/IEC JTC1/SC27, Common Criteria for Information Technology Security Evaluation - Part 1: Introduction and general model, CCMB 2006-09-01, 2006.

[22] Insight Consulting, The Logic behind CRAMM'sAssessment of Measures of Risk and Determination of Appropriate Countermeasures, 2005.

[23] B. Karabacak, I. Sogukpinar, ISRAM: information security risk analysis method, Computers & Security, 24 (2005), 147-159.

[24] C. J. Alberts, A. J. Dorofee, OCTAVE Criteria Version 2.0, CMU/SEI-2001-TR-016. ESC-TR-2001-016, 2001.

[25] M. Howard, D. C. LeBlanc, Writing Secure Code, Microsoft Press, Redmond, WA, USA, 2002.

[26] S. E. Schechter, Computer Security Strength & Risk: A Quantitative Approach, PhD Thesis, Harvard University, Boston, USA, 2004.

[27] B. Schneier., Attack trees: Modeling security threats, Dr. Dobb's Journal , December, 1999.

[28] S. Jha, O. Sheyner, J. Wing, Two formal analyses of attack graphs, in Proceedings of the 15th Computer Security Foundation Workshop, pp. 49-63, 2002.

[29] P. Pamula, P. Ammann, A.Jajodia, V. Swarup, A weakest-adversary security metric for network configuration security analysis, in Conference on Computer and Communications Security, Proceedings of the 2nd ACM workshop on Quality of protection, pp. 31 - 38 ,2006.

[30] P. Ammann, D. Wijesekera, S. Kaushik, Scalable, graph-based network vulnerability analysis, in Proceedings of 9th ACM Conference on Computer and Communications Security, pp. 217 - 224, 2002.

[31] S. Jajodia, S. Noel, B. O'Berry, Topological analysis of network attack vulnerability, in Managing Cyber Threats: Issues, Approaches and Challanges, chapter 5. Kluwer Academic Publisher, V. Kumar, J. Srivastava, and A. Lazarevic, Eds. Springer US, 2003, pp. 247-266.

[32] T. Tidwel, R. Larson, K. Fitch, J. Hale, Modeling Internet attacks, in IEEE Workshop on Information Assurance and Security, pp 54-59, West Point, NY, USA, 2001.

[33] C. Phillips, L. P. Swiler, A graph-based system for network-vulnerability analysis, in Proceedings of the 1998 workshop on New security paradigms, pp. 17-79, 1998.

[34] X. Ou, W. F. Boyer, M. A. McQueen., A Scalable Approach to Attack Graph Generation, in Proceedings of the 13th ACM Conference on Computer and Communications Security, pp. 336-345, 2006.

[35] Y. Liu, M. Hong, Network vulnerability assessment using Bayesian networks, in proceedings of SPIE, , pp. 61-71Orlando, Florida, USA, 2005.

[36] W. J. Caelli, D. Longley, A. B. Tickle, A methodology for describing information and physical security architectures, in Proceedings of the IFIP TC11, Eigth International Conference on Information Security, vol. A-15, pp. 277–296, Singapore, 1992.

[37] A. Anderson, D. Longley, L. F. Kwok, Security modelling for organisations, in Proceedings of the 2nd ACM Conference on Computer and communications security, pp. 241-250, Fairfax, Virginia, United States, 1994.

[38] S. Bistarelli, F. Fioravanti., P. Peretti., Defense trees for economic evaluation of security investments, in in proceedings of The International Conference on Availability, Reliability and Security, pp. 416-423, Vienna, Austria, 2006.

[39] S. Bistarelli, M. Dall'Aglio, P. Peretti, Strategic games on defense trees, in Formal Aspects in Security and Trust. Springer Berlin / Heidelberg, 2007, pp. 1-15.

[40] S. Bistarelli, F. Fioravanti, P. Peretti, Using CP-nets as a Guide for Countermeasure Selection, in Proceedings of the ACM symposium on Applied computing, pp. 300 – 304, Seoul, Korea, 2007.

[41] O. Sheyner, J. Wing, Tools for Generating and Analyzing Attack Graphs, in Formal Methods for Components and Objects. Springer Berlin / Heidelberg, 2004, pp. 344-371.

[42] L. P. Swiler, C. Phillips, D. Ellis, S. Chakerian, Computer-attack graph generation tool, DARPA Information Survivability Conference & Exposition II, 2001. DISCEX '01. Proceedings , vol.2, pp.307-321, 2001.

[43] R. W. Ritchey, P. Ammann, Using model checking to analyze network vulnerabilities, in Proceedings of the IEEE Symposium on Security and Privacy, pp. 156–165, 2001.

[44] T. Sommestad, M. Ekstedt, P. Johnson, Cyber Security Risks Assessment with Bayesian Defense Graphs and Architectural Models, in 42nd Hawaii International Conference on System Sciences, pp. 1-10, Hawaii, 2009.

[45] F. V. Jensen, An Introduction to Bayesian Networks, New York: Springer-Verlag, 1996.

[46] W. H. Hsu, R. Joehanes, Relational Decision Networks, in Working Notes of the ICML-2004 Workshop on Statistical Relational Learning and Connections to Other Fields, pp. 61-67, Banff, Canada, 2004.

[47] R. Shachter, Evaluating influence diagrams, Operations Research, 34 (1986) 871-882.

[48] W. Schwartau, Time-based security explained: Provable security models and formulas for the practitioner and vendor, Computers & Security, 17 (1998), 693-714.

[49] M. Ekstedt, U. Franke, P. Johnson, R. Lagerström, T. Sommestad, J. Ullberg, M. Buschle, A Tool for Enterprise Architecture Analysis of Maintainability. In Proceedings of the 2009 European Conference on Software Maintenance and Reengineering, 327-328,Washington, DC, USA, 2009,

[50] P. Giorgini, F. Massacci, J. Mylopoulos, N. Zannone, Modeling security requirements through ownership, permission and delegation, in 13th IEEE International Conference on Requirements Engineering, 167- 176, Paris, France, 2005.

[51] J. Jürjens, P. Shabalin, Automated Verification of UMLsec Models for Security Requirements, 2004 - The Unified Modelling Language, pp. 365-379, Springer Berlin/Heidelberg, 2004.

[52] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos, Tropos: An Agent-Oriented Software Development, Autonomous Agents and Multi-Agent Systems , 8(2004), 203-236

[53] SMILE. Decision System Laboratory, University of Pittsburgh, http://genie.sis.pitt.edu/

[54] C.Yuan M.J Druzdzel. Mathematical and Computer Modelling, 43(2006), 1189-1207.