Reusing Risk Analysis Results

An Extension for the CORAS Risk Analysis Method

Johannes Viehmann
Fraunhofer Institut FOKUS (MOTION)
Kaiserin-Augusta-Allee 31
D-10589 Berlin, Germany
Johannes.Viehmann@Fokus.Fraunhofer.de

Abstract — **This paper shows how the results of CORAS risk analysis can be reused and combined. It introduces new models, diagram types and procedures as an extension of the CORAS method. Taking risk analysis artifacts generated for the individual base components as input, probability values for unwanted incidents of complex systems can be calculated if the relations between these artifacts are modeled correctly. Initially developed for the S-Network, a trustworthy repository, this extension is predestined for analyzing large scale systems consisting of heterogeneous components, which no single analyst team could handle.**

Risk analysis, CORAS, fault tree analysis, trust, S-Network

I. Introduction

In many applications in varying market sectors, including eCommerce, eGovernment, and eHealth, perfect security cannot be achieved. Trust allows people to use such applications though there are remaining risks. Before taking risks, it is reasonable to carefully analyze the chances, the potential benefits and the potential losses. Those offering security critical applications or services can use risk analyzes to treat potential weaknesses in their products. Communicating the identified remaining risks honestly can be important to create trust. However, risk analyzes might be difficult and expensive. This paper introduces new concepts to reuse and combine results of the CORAS method for risk analysis.

These concepts were created during the still ongoing development process for a large-scale trustworthy repository called the *S-Network* (http://Surn.net). The *S-Network* is going to provide guarantees for the long term preservation and for the permanent secure non-repudiation accessibility of its content. Requiring all users to agree on a user contract, the *S-Network* will offer legal validity for its content, including verifiable metadata values (e. g. who stored what and when) with standardized legal implications for all participants. The *S-Network* is intended to become a universal platform for applications that have most stringent requirements, e.g. fair contract signing. Indeed, it must be resistant to both manipulation attempts and censorship. However, since it will not be possible to develop a perfectly secure solution, remaining risks have to be analyzed and communicated in order to create trust in the *S-Network*.

Instead of the *S-Network* itself, a service for generating time-stamps is analyzed in the scope of this paper as a more compact example. It is a relatively small, but often security critical component. For instance, the *S-Network* requires reliable time-stamps for its publications and deposits because it must be possible to determine when these were stored. This smaller example is used to argue why the suggested extension of CORAS was created for the development of the *S-Network*.

II. Notation, Problems and State of the Art

A. Risk analysis with FTA, FME(C)A and probability theory

Fault tree analysis (FTA) [15] is widely used in the process of risk analysis for critical systems like airplanes or nuclear power plants and hence well-studied [6]. It is a deductive, top-down approach to study how faults can be triggered by sets of other faults. FTA considering temporal effects is called dynamic FTA. Analyzing potential failure paths, FTA makes it possible to determine the probability that a single top level fault occurs. All possible paths have to be taken into consideration by the analysts. Starting at the top level fault, it might be very difficult to recognize all initiating faults that could somehow cause the top level fault.

In contrast, failure modes and effects (and criticality) analysis FME(C)A [4] is commonly used as an inductive, bottom-up approach. FME(C)A is better for identifying initial failures than FTA, but not for getting a complete analysis of a complex failure. It can be beneficial to do both, FME(C)A and FTA because they have complementary strengths. In [1], a combination of both is suggested as "Bouncing Failure Analysis (BFA): The Unified FTA-FMEA Methodology".

Based upon [12], for calculating probability values, the following notations and equations/formulas will be used in this paper: The probability of some incident X is noted as $P(X)$. The conditional probability for incident X given that it is known that incident Y occurs is noted as $P(X \mid Y)$. The probability $P(X \cap Y)$ that both incidents X and Y occur can be calculated with $P(Y)$ and $P(X \mid Y)$:

$$P(X \cap Y) = P(X|Y) * P(Y) \quad (1)$$

The probability $P(X \cup Y)$ that at least one of two incidents X, Y occurs is:

$$P(X \cup Y) = P(X) + P(Y) - P(X \cap Y) \quad (2)$$

If X and Y are statistically independent (i.e. $P(X \mid Y) = P(X)$ and $P(Y \mid X) = P(Y)$), then:

Integrated Graduate Program H-C3 (IGP H-C3, http://www.h-c3.org) of the Berlin Institute of Technology (Technische Universität Berlin).

Going to be presented at PASSAT 2012 in Amsterdam and published in the conference proceedings (http://www.ieee.org),

$$P(X \cap Y) = P(X) * P(Y) \quad (3)$$

$$\mathrm{P}(X \cup Y) = \mathrm{P}(X) + \mathrm{P}(Y) - P(X) * P(Y) \quad (4)$$

If P(X | Y) = 1 and P(Y | X) = 1, then:

$$P(X \cap Y) = P(X \cup Y) = P(X) = P(Y) \quad (5)$$

The probability value V for an incident that has to be triggered by at least *threshold* Ψ of n statistically independent incidents each having the probability p can be calculated using the following binomial formula [13]:

$$\mathrm{V} = \sum_{k=\Psi}^{n} \binom{n}{k} * p^k * (1-p)^{n-k} \quad (6)$$

Multiple algorithms are known for calculations in FTA – including the binary decision diagram based (BDD) algorithm presented in [11] and DIFtree [7] using both BDD and Markov chains. Various software tools support FTA, e.g. Galileo [5].

B. *Risk analysis with the CORAS method*

In contrast to the pure failure analytic methods FTA/ FME(C)A, the model based CORAS method [8] supports the entire process of risk analysis "from asset identification to risk treatment" [10].

The CORAS method consists of eight steps. Following this guided step by step procedure, it is possible to identify, analyze and evaluate assets, threats, risks and possible treatments. During that process, different types of diagrams with intuitively understandable graphic symbols are generated as results. CORAS diagrams can be translated to English paragraphs [3]. Besides the completeness, the easy comprehensibility of the CORAS artifacts makes the CORAS method a good choice for analyzing the risks of the *S-Network* because communicating the risks is essential for creating trust in the *S-Network*.

In this paper, the CORAS terminology will be used. Threat diagrams and risk diagrams will be used and extended.

C. *CORAS risk analysis complexity and difficulty*

Many computer programs and services are composed of different components – developed, produced and operated by different entities. There is no need to reinvent the wheel or recreate things that already exist. Often, it seems that the internals of existing components do not have to be studied in detail to be able to utilize them because public interfaces and their documentation typically describe the functionality.

However, each single component might eventually have certain risks. For the risk analysis of an entire complex system, to identify the risks inherited from the components it consists of, it would be necessary to get a deep understanding about the internals of these base components. Probably only the producers or operators of each base component will have the required knowledge. Additionally, it would be inefficient to analyze the same base component which is used in many systems over and over again for each system containing that component.

If risk analysis results for individual components were reusable and if they could be composed along with the components to get the risks of complex systems consisting of these components, there would be no need to analyze them again and again. In [2] "Dependent CORAS Diagrams" are suggested to deal with dependencies of different components. But these diagrams are only appropriate to hide some complexity from the "context scenario". Hence, in [10] chapter 16, "Dependent CORAS" is only mentioned for dealing with assumptions about the environment, which could then be replaced with risk analysis results about the environment. There is not yet a satisfying solution for composing CORAS risk analysis results in not trivial ways.

III. Composition of Risk Analysis Artifacts

The idea presented here to make the risk analysis for complex systems more feasible is to use the conventional CORAS method only for the relatively small individual components the system consists of. Composing the resulting artifacts of such analysis along with the combination of the components should allow to detect and to evaluate the risks of the complex system. Combining components, their risks could be reduced; increased or even new risks might arise.

In the scope of this paper, the conventional CORAS risk analysis process will not be presented in detail. Instead, just some results are given. The risk analysis artifacts shown here are exemplary excerpts – they are not meant to be complete. Figure 1 shows a *threat diagram* for the exemplary time-stamp service that will be used as the base for all further risk analysis throughout this paper.

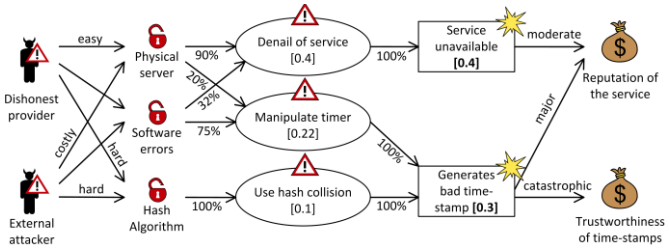

Figure 1. CORAS threat diagram

In step four of the CORAS method the scales for expressing likelihoods, consequences and the functions to calculate risk values are defined by those who do the risk analysis. This freedom makes it eventually difficult to reuse results of the risk analysis for different components if they use deviating scales and risk functions. Eventually, it might be necessary to define and apply proper conversion functions.

In this example, absolute likelihoods of threat scenarios and unwanted incidents to occur within a time period of ten years are noted within square brackets as probability values according to the Kolmogorov axioms [9], i.e. as real numbers between zero (will not happen) and one (will definitely happen). Relative likelihoods on relations are noted as percent values. For example if someone exploits "software errors", Figure 1 indicates that there is a relative likelihood of 32% that this is a denial of service attack and there is a relative likelihood of 75% that this attack manipulates the timer. Note that attackers can try to do both simultaneously with a single attack, so the sum of the relative likelihoods for the consequences may be above 100%.

A. *Creating reusable threat interfaces for components*

A *threat interface* describes how an individual component could be influenced by the unwanted incidents of other components and how it could itself affect the security of other components or the entire system. It should hide internal details. But it must be detailed enough to model and to evaluate the threats of a complex system composed of multiple components.

A *threat interface* consists of a descriptive name for the component and three lists: The first list contains vulnerabilities that are exposed to other components. The second list contains unwanted incidents that might be a threat for other components or for the entire system. The third list contains directed relations, each having a vulnerability from the first list as starting point and an unwanted incident of the second list as end point.

The threat diagrams created in step five and six of the conventional CORAS method contain all the information required to define a *threat interface*: These diagrams give a detailed picture by distinguishing between vulnerabilities, threat scenarios and unwanted incidents.

The vulnerabilities and the unwanted incidents from the threat diagram can directly be used within the *threat interface*. The threat scenarios are somehow internal. They are hidden in the *threat interface*: Each relation path in the threat diagram leading from a vulnerability to a threat scenario and further to an unwanted incident is replaced in the *threat interface* by a direct relation between the corresponding vulnerability and the corresponding unwanted incident. The relative likelihood values of the replaced relations are multiplied to get the relative likelihood for each new direct relation.

Threat interfaces for components have a graphic representation as a box with vulnerabilities on the left hand side and the unwanted incidents on the right hand side. Arrows with dashed lines represent the relations. Relative likelihood values are written under the arrows. The *threat interface* for the time-stamp service is shown in Figure 2.

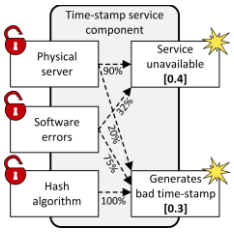

Figure 2. *Threat interface*

B. *Threat composition diagram*

In the example, it is more likely that the time-stamp service becomes unavailable, but the threat diagram (Figure 1) shows that if the service generates a bad (i.e. wrong or weak) time-stamp, the consequences are expected to be more serious. In step seven and eight of the conventional CORAS method, both unwanted incidents are therefore evaluated as high risks and both should be treated.

One possible treatment to improve the availability of time-stamp generation is to use multiple time-stamp services. If there are two alternative time-stamp services $\{A, B\}$ and if it is enough that just one of them is accessible, then the combined service $A \vee B$ would still be accessible even if one base service becomes unavailable. Client applications can directly contact one of the two services $\{A, B\}$. Hence, the combined service $A \vee B$ does not have to be implemented. It can be just a logical service. Instead of doing a complete conventional CORAS risk analysis for the logical service $A \vee B$, the idea is to make a *threat composition* with the *threat interfaces* for the base services. Therefore, the *threat composition diagram* is introduced:

The *threat composition diagram* consists of two layers. The *component layer* contains information about how the base components themselves are combined to a complex component. The second layer contains information about the vulnerabilities and unwanted incidents identified for each individual component and about how these could affect one another. That layer is called the *directed graph of consequences*.

In a *threat composition diagram*, each individual component is represented by its *threat interface*. The relations between the components are modeled on the *component layer* as relations between the entire *threat interfaces* using arrows with dotted lines. If a simple arrow is not enough to make the relation understandable, *description boxes* may be used to informally explain relations. For the relations between the components in the *threat composition diagram* in Figure 3 there is a *description box* on the side of the *threat interface* for time-stamp service $A \vee B$ having the value "≥1". This means that the new combined time-stamp service $A \vee B$ relies on the output of at least one of its base services $\{A, B\}$.

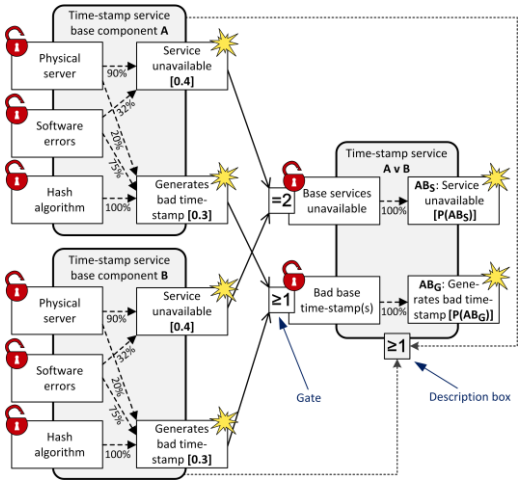

Figure 3. Threat composition diagram with three components

For each component, the *threat interface* is generated from a threat diagram produced in a conventional CORAS risk analysis processes. If a component is composed of other base components, that analysis should not go into the details of the base components. Instead, vulnerabilities corresponding to the unwanted incidents of the base components are identified. Numeric values for the probability of unwanted incidents which could be triggered by unwanted incidents of the base components do not have to be estimated in the conventional CORAS analysis process. These values can be calculated using the *directed graph of consequences*.

While the *threat interfaces* themselves become a part of the *component layer* in a *threat composition diagram*, their vulner-

abilities and unwanted incidents become nodes in the *directed graph of consequences*. The internal relations of threat interfaces become edges in the *directed graph of consequences*. Additionally, if an unwanted incident of some *threat interface* could affect a vulnerability of another *threat interface*, that relation is modeled as another edge in the *directed graph of consequences*. In the *threat composition diagrams*, the graphic representation for such a *trigger relation* is an arrow having a continuous line.

Gates can be used to express complex *trigger relations*: For example, multiple unwanted incidents might be required to actually affect a specific vulnerability. Graphically, a *gate* is represented by a small square with a label representing its function. In the example (Figure 3), the combined time-stamp service $A \vee B$ becomes only unavailable if service A and service B are unavailable. That is why the *gate* on the side of the "base services unavailable" vulnerability has the label "=2". But if just one of base service A or base service B produces a bad time-stamp, this will cause the combined service $A \vee B$ to produce a bad time-stamp, too. Therefore, the *gate* on the side of the "bad base time stamp(s)" vulnerability has the label "≥1".

The *directed graph of consequences* with its *gates* is similar to a fault tree and it will allow doing similar calculations of probability values. The unwanted incidents correspond to faults in a fault tree. But in contrast to a fault tree, the *directed graph of consequences* does not have to be a tree. It can have multiple top level incidents, for example. With the vulnerabilities, the *directed graph of consequences* contains significantly more information than a fault tree. Furthermore, it is always integrated in a *threat composition diagram*, which contains information about the components and their combination.

In particular, vulnerabilities are important for identifying potential statistical dependencies not yet modeled in a *threat composition diagram*. Knowing the statistical dependencies is essential to calculate probability values accurately.

If two unwanted incidents can be triggered by the same unwanted incident (i.e. they have a common trigger unwanted incident in the *directed graph of consequences*) then they are definitely somehow statistically dependent. In the *threat composition diagram* shown in Figure 3, there are no such obvious dependencies. Unfortunately, if there is no common trigger in the *threat composition diagram*, this does not necessarily mean that the unwanted incidents are statistically independent.

For statistical independency, there are no such simple criteria. Eventually, the *threat composition diagram* is not fine grained enough and a more detailed analysis is required. Can there possibly be some common trigger incidents that were not yet modeled? Looking at the incidents or faults without knowing further details about the components, it is impossible to answer that question. Vulnerabilities are exactly the missing information: they tell the analysts directly how a component could be affected. A careful look at the vulnerabilities especially of the base components is crucial. In Figure 3 the two base services have vulnerabilities with identical labels. That is a clear indicator that probably there could be some dependencies. In such a case, additional *threat interfaces* for more base components have to be added and their relations must be modeled. Thereby, the *threat composition diagram* gets finer grained.

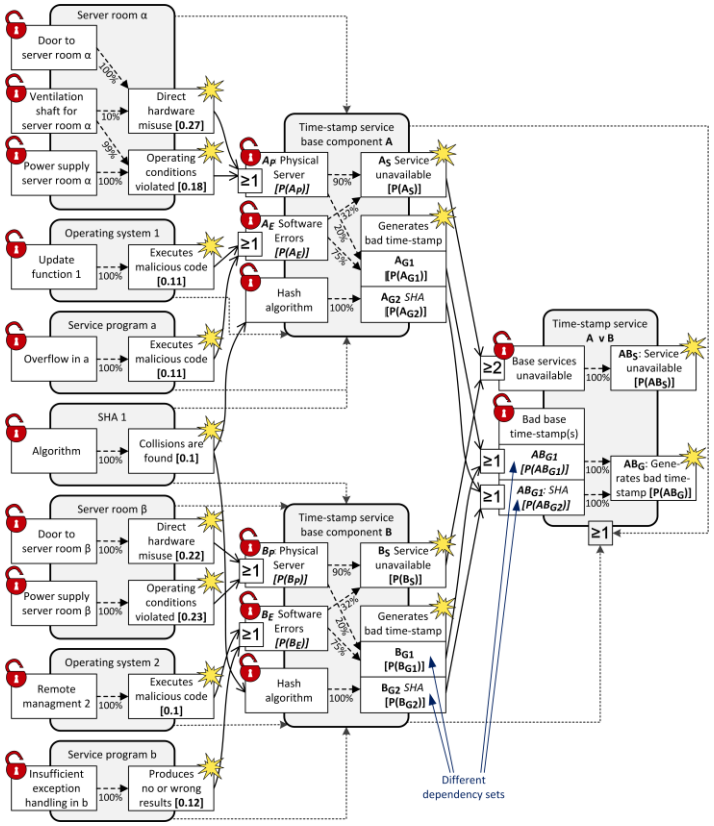

Figure 4. Threat composition diagram with additional base components

For the time-stamp service base components A and B, there are several base components taken into consideration here: the server room, the operating system, the service program and a hash algorithm. Figure 4 shows the *threat composition diagram* with the *threat interfaces* for these components.

In the example, both base time-stamp services rely on the SHA-1 algorithm. If that algorithm is broken and collisions can be found easily, both services will produce weak time-stamps. In the *directed graph of consequences* the "generates bad time-stamp" incident of service A and the "generates bad time-stamp" incident of service B can both be triggered by the same unwanted incident "collisions are found" of the "SHA-1" component. Clearly, having a common trigger node, they are statistically dependent. In a fault tree, it would be necessary to model two separate nodes having the same name for the two triggers. Else, it would not be a tree. The *directed graph of consequences* can model dependencies more directly end intuitively.

The "generates bad time-stamp" incidents of the base services can also be triggered by other incidents obviously not having the same dependency. The probability values for unwanted incidents caused by triggers of different dependencies must be kept separately to make a correct probability value calculation possible. Therefore, an unwanted incident can have multiple different *dependency sets*, each representing only those incidents caused by triggers that have the same dependency throughout the entire *directed graph of consequences*. In the *threat composition diagram*, the dependency sets are represented as rows in the box representing the incident. *Dependen-*

cy sets should have a description indicating the cause of dependency if there is any dependency. In the example, the description for the *dependency set* triggered by the "collisions are found" incident is *SHA*.

Vulnerabilities can be affected by unwanted incidents having multiple different dependencies. In order to support the correct calculation of probability values for the possible consequences, information about different *dependency sets* of the "input" unwanted incidents must be preserved in vulnerabilities. Therefore, the vulnerabilities in the *threat composition diagram* can have multiple rows representing different *dependency sets*, too. Relations between an unwanted incident and a vulnerability (or vice versa) both having the same *dependency sets* are modeled directly between the dependency sets. That way, probability values for different *dependency sets* can be propagated through the *directed graph of consequences* without mixing them up. In the example, the "bad base time-stamp(s)" vulnerability of time-stamp service $A \vee B$ preserves the *dependency sets* of the incidents which can affect it.

For the *dependency set* named *SHA*, the kind of dependency is visible in the *directed graph of consequences* in the common trigger "collisions are found". For the other *dependency set*, Figure 4 does not show any common triggers. Does that mean that these parts of the "generates bad time-stamp" incidents of the two base services are statistically independent? The "service unavailable" unwanted incidents of base time-stamp service *A* and of base time-stamp service *B* do not have a common trigger, too. The graph shows no dependency between them. Is the diagram fine grained enough to decide whether they are statistically independent?

The only way to figure that out is to look carefully at the vulnerabilities. For all the vulnerabilities that could eventually have a common trigger, a closer look at a finer grained *threat composition diagram* is required.

One of the vulnerabilities by which both unwanted incidents of time-stamp service *A* could be affected is called "power supply server room α". For time-stamp service *B*, there is a similar vulnerability called "power supply server room β". If both server rooms are connected to the same electricity network with the same power plants, then these can definitely be affected by the same unwanted incidents. A closer look with more fine grained components and risk interfaces is required here to decide about statistical independency. Figure 5 shows a detailed *threat composition diagram* excerpt just for the two power supplies.

The two incidents *X* (EG1 fails to produce power) and *Y* (EG2 fails to produce power) are statistically dependent. They are an example for mutual dependency, they can affect each other. If both electric generators EG1 and EG2 work within normal parameters, each has to produce 10 KW. If one of these generators fails, then the other generator has to produce up to 20 KW (the maximum capacity). A generator which has to produce 20 KW needs more cooling. It becomes more likely that it will overheat. Though the cooling systems themselves are independent – their vulnerabilities will not be affected by the same incident – the failure of one generator increases the likelihood that the other generator will overheat.

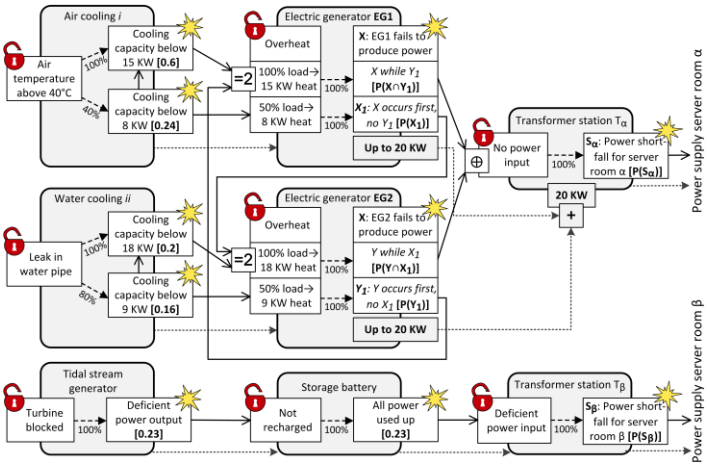

Figure 5. Threat composition diagram for power supply

As long as EG2 works, EG1 needs to produce only 10 KW and for air cooling *i* a cooling capacity of 8 KW is sufficient. The probability $P(X_1)$ that EG1 will fail under such conditions is 0.24. The conditional probability $P(Y \mid X_1)$ that EG2 will fail if EG1 has already failed is 0.2 because that is the probability that the cooling capacity of "water cooling ii" will drop below 18 KW. Once EG2 has to produce the entire 20 KW, it needs at least that cooling capacity.

With the formula from equation (1), it is possible to calculate the probability that both electric generators will fail if EG1 fails first: $P(Y \cap X_1) = 0.048$.

The probability $P(X \cap Y_1)$ that both electric generators will fail if EG2 fails first can be calculated the same way. $P(Y_1)$ is 0.16, $P(X \mid Y_1)$ is 0.6 and $P(X \cap Y_1) = P(X \mid Y_1) * P(Y_1) = 0.096$.

Only if both generators fail at the same time, there will not be enough power for server room α. Each of the two incidents $X \cap Y_1$ and $Y \cap X_1$ alone can trigger the "power shortfall for server room α" incident S_α. Figure 5 indicates that if X_1 occurs, Y_1 will not occur and vice versa. X_1 and Y_1 are mutually exclusive. Therefore, $X \cap Y_1$ and $Y \cap X_1$ are mutually exclusive, too, i.e. $P((X \cap Y_1) \cap (Y \cap X_1)) = 0$. The probability $P(S_\alpha)$ that the power supply for server room α fails can be calculated using the formula from equation (2): $P(S_\alpha) = 0.144$.

This dependency affects transformer station T_α and server room α, but neither transformer station T_β nor server room β. The *threat composition diagram* in Figure 5 shows no dependencies between the two transformer stations T_α and T_β. It is detailed enough to see that the base vulnerabilities do not have any common trigger incidents with a relevant likelihood. The two "power shortfall" incidents are statistically independent. The power supply for server room α is indeed completely separated from the power supply for server room β. There are no statistical dependencies between the two power supplies that have to be taken into consideration in order to calculate probability values correctly for the *threat composition diagram* given in Figure 4.

Once this top-down analysis is completed and the *threat composition diagram* is detailed enough to decide about statistical independencies, a bottom-up analysis is required to propagate any identified new dependencies with the help of *de-*

pendency sets throughout the entire *directed graph of consequences*. Having finer grained components, analysts doing the bottom-up analysis will eventually identify some vulnerabilities and incidents in higher level components that have been overlooked before. Eventually, further bouncing analysis going multiple times top-down and bottom-up might be necessary to get a complete picture. Because the *directed graph of consequences* can have multiple top level incidents, it is possible to do this bouncing analysis without changing the model. In a fault tree, this would not be possible.

For the time-stamp service example, in the scope of this paper, details of the finer grained analysis for other components than the power supplies are omitted. Instead, just the result is given: The fine grained component analysis reveals no additional statistical dependencies between the unwanted incidents of the two base time-stamp services.

Having a *threat composition diagram* with complete information about the dependencies and absolute probability values at least for all initial unwanted incidents, it becomes possible to calculate the missing probability values. For each top level incident, this calculation works like in a fault tree.

Figure 4 shows, that the incident AB_S (i.e. the combined time-stamp service $A \vee B$ becomes unavailable) occurs only if both "service unavailable" incidents $\{A_S, B_S\}$ of the two base services $\{A, B\}$ occur, i.e. $P(AB_S) = P(A_S \cap B_S)$. The analysis shows that $\{A_S, B_S\}$ are statistically independent. Hence, it is possible to apply the formula from equation (3) and $P(AB_S)$ is simply $P(A_S) * P(B_S)$.

The incident A_S can be triggered by unwanted incidents that affect the vulnerability physical server (A_P) or by unwanted incidents that affect the vulnerability software errors (A_E). There is no statistical dependency between any incident affecting A_P and any incident affecting A_E. If A_P is affected by some incident, this will trigger in 90% of all cases A_S. If A_E is affected by some incident, this will trigger in 32% of all cases A_S. Therefore:

$$P(A_S) = 90\% * P(A_P) + 32\% * P(A_E) - 90\% * P(A_P) * 32\% * P(A_E)$$

Both $P(A_P)$ and $P(A_E)$ can be calculated using the formula from equation (4) and the absolute probability values of the incidents that affect them as parameters. The results are: $P(A_P) = 0.4, P(A_E) = 0.21, P(A_S) = 0.4$. $P(B_S)$ can be calculated the same way and has a numeric value of 0.4, too. Finally, it is possible to calculate $P(AB_S)$, which is 0.16.

The combined time-stamp service $A \vee B$ will produce bad time-stamps if at least one of the two base services produces a bad time-stamp. For the unwanted incident AB_G there are trigger incidents belonging to two different dependency sets. AB_{G1} represents the vulnerability bad base time-stamp(s) being affected by statistically independent incidents. AB_{G2} represents the vulnerability bad base time-stamp(s) being affected by incidents depending on the SHA-1 collisions found incident. Any incident that could affect AB_{G1} is statistical independent from any incident that could affect AB_{G2}. To calculate $P(AB_G)$, it is possible to apply the formula from equation (4) with $P(AB_{G1})$ and $P(AB_{G2})$ as parameters. Using the same formula several times and applying the relative likelihoods correctly, it is possible to calculate $P(AB_{G1})$. $P(AB_{G2})$ can be trivially calculated using the formula from equation (5). The numeric values are: $P(AB_{G1}) = 0.39$, $P(AB_{G2}) = 0.1$, $P(AB_G) = 0.45$.

The probability that a bad time stamp will be generated has been increased by taking one result of two different base services. To improve both, the availability and the correctness, it would probably be a good idea to use three different base services and to require at least two of them to confirm the same time-stamp value.

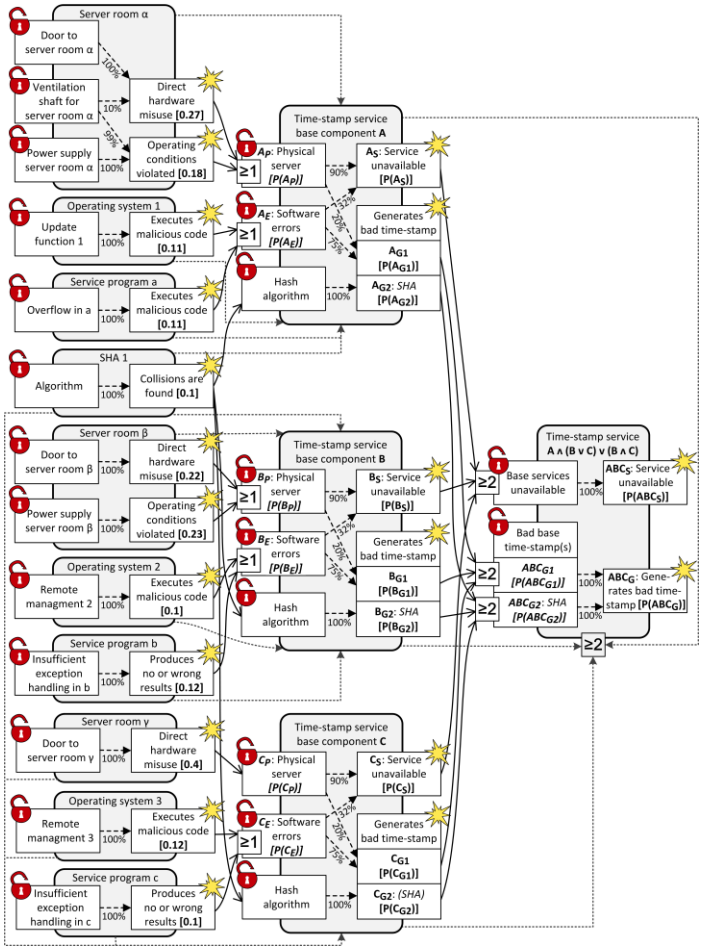

Figure 6. Threat composition diagram with three base services

In the example shown in Figure 6, given that all relevant statistical dependencies that are shown in the diagram, it is possible to apply the formula from equation (6) to calculate $P(ABC_S)$ and $P(ABC_{G1})$. The numeric results are: $P(ABC_S) = 0.352$ and $P(ABC_G) = 0.212$. For the logical time-stamp service $A \wedge (B \vee C) \vee (B \wedge C)$, both the availability and the correctness are improved.

In the design of the *S-Network*, such or even higher redundancy is widely used for many different aspects including bit sequence preservation, meta-data generation and access control. Careful analysis for dependencies as shown here is required to estimate the effect of redundancy on the probability values for the unwanted incidents. In complex systems, it is not always obvious whether it is better to use identical or divergent technologies and implementations. *Threat composition diagrams* are really helpful in this analysis for the *S-Network*.

C. *Composition with external threats and assets*

Just looking at the *threat interfaces* of the components might eventually not be enough. External threats (especially human threats) identified for different components in separate threat diagrams could probably interact with one another. There could be new combined threats, resulting in different dependencies of unwanted incidents. Hence, the probability values can only be calculated correctly if the potential combinations of threats are modeled and composed correctly.

A real time-stamp service typically has a provider who is responsible for the operation of the time-stamp service. The provider owns the server room and he has the key offering easy access to the physical server his service runs on. A dishonest provider could use his privileged access to manipulate the service he is responsible for. Therefore, the threat diagram shown in Figure 1 contains a human threat "dishonest provider".

If a logical time-stamp service is composed of multiple real time-stamp services, no single entity should provide more than one of the real base services. Consequently, no single entity would have easy access to more than one of the servers used for a base time-stamp services (Figure 7). Manipulating just one base service would then not be enough to manipulate the logical combined service $A \wedge (B \vee C) \vee (B \wedge C)$. The easy access to a single server of a base service becomes less critical.

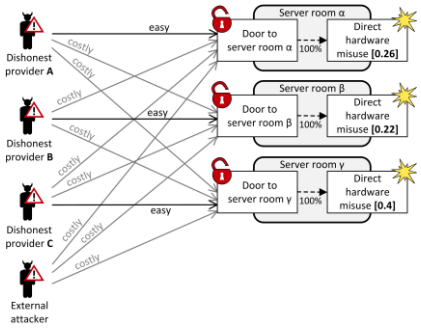

Figure 7. Difficulties to physically access the server rooms for the different human threats

However, this does not mean that having multiple services with different providers is automatically more secure: Two providers could agree to cooperate with one another to cheat successfully. For those who collaborate, it does not matter that each of them only has the key to exactly one single server room: Two or more allied providers working together do have at least two keys and therefore easy access to at least two different server rooms. Access to two server rooms is enough to manipulate the combined logical service successfully.

In the *threat composition diagram*, potentially manipulative coalitions can be represented by *threat interfaces* as shown in Figure 8. Each manipulative collaboration incident of these interfaces can trigger direct hardware misuse incidents in two or more different server rooms. Incidents triggered by the same initial manipulative collaboration incident are statistically dependent. The different dependencies have to be modeled as separate dependency sets. In the example, each direct hardware misuse incident has four different dependency sets. These different dependencies have to be propagated forward through the directed graph of consequences as shown in Figure 9.

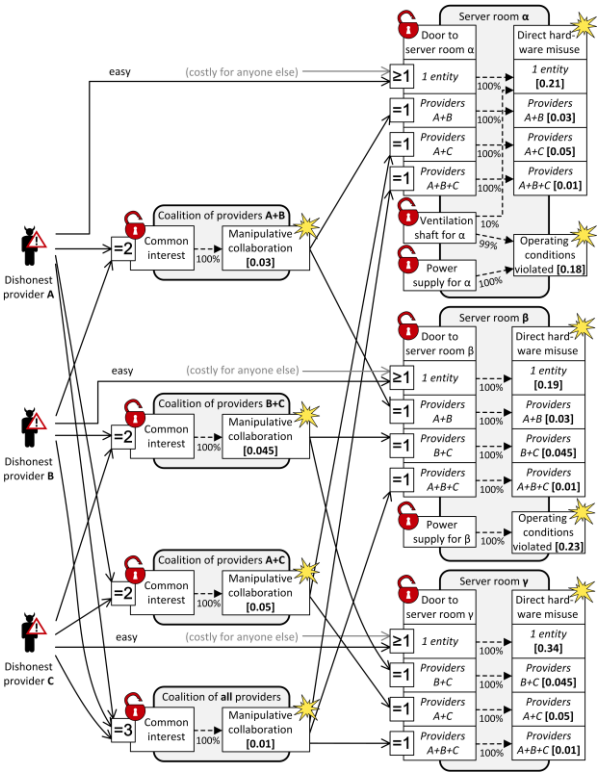

Figure 8. Threat composition diagram with coalitions (excerpt 1)

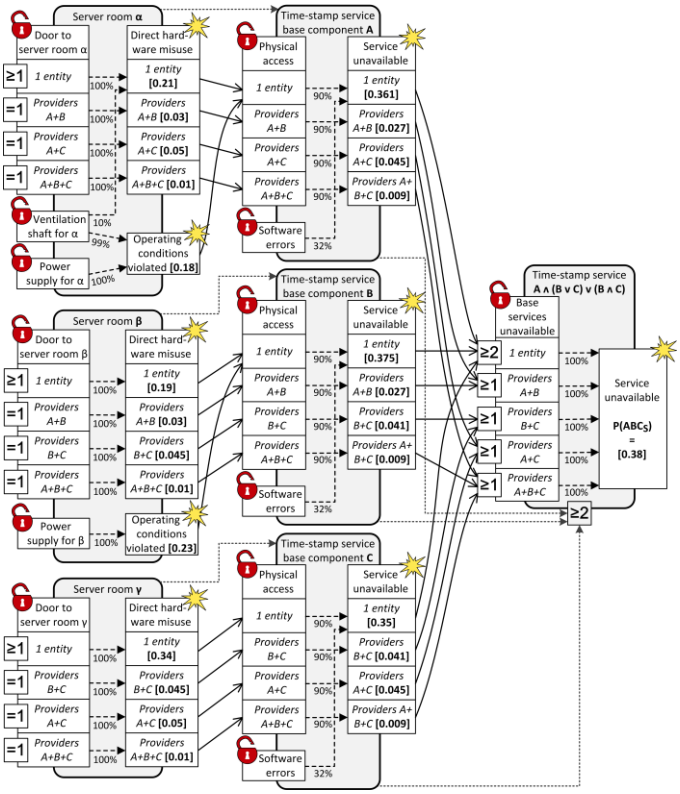

Figure 9. Threat composition diagram with coalitions (excerpt 2, containing only the service unavailable top level incident, with probability value results)

In general, all threats identified in the threat diagram for any individual involved component have to be added to the

threat composition diagram. Each threat should appear only once. For each relation from a threat *K'* to some vulnerability *M'* in the threat diagram of an individual component it is necessary to make sure that there is a relation between the corresponding threat *K* and the vulnerability *M* in the *threat composition diagram*, too. Eventually, it is not necessary to insert a direct relation: If there is a relation leading from the threat *K* to another vulnerability *N*, and if there is a path between *N* and *M* in the *directed graph of consequences* having a relative likelihood of 100%, then this indirect relation is sufficient.

For the actual composition analysis process of external threats (e.g. the process of finding potentially harmful coalitions of human threats), there is no simple algorithm. Those threats that can affect some of the involved components, but not all of them (at least not in the same way) are candidates for composition analysis. If there can be any interaction between these threats which could affect the new composed system, these interactions and the resulting dependencies must be modeled in the *threat composition diagram*.

The *threat composition diagram* is not yet complete until the consequences of unwanted incidents for the assets are taken into consideration, too. All consequences and assets identified in the threat diagrams for individual components have to be included in the *threat composition diagram*. Let *T'* be an asset identified in the threat diagram for the component *D'*. If there is not yet an asset *T* corresponding to *T'* in the *threat composition diagram*, then *T* must be added.

For each unwanted incident *E'* identified for component *D'* that has the consequence *Q'* for *T'*, it is necessary to make sure that this consequence is also modeled correctly as a consequence relation *Q* between E (i.e. the incident corresponding to *E'* in the *threat interface* for *D'*) and *T* in the *threat composition diagram*. The consequence value of *Q'* is assigned to *Q*.

Threats and their influence relations are added to the *threat composition diagram* to support the analysis of dependencies and to enable the correct calculation of probability values. Consequences and assets are basically added to the *threat composition diagram* because these are required for the further steps in the risk analysis process.

For the *S-Network*, manipulative coalitions have been identified as a major threat using the risk analysis method shown here. Having effective treatments that prevent such collaboration of human threats is considered to be crucial. A detailed analysis of the coalition threats with measures for preventing these is presented in [14].

IV. Deriving and Comparing Risks

For identifying and evaluating risks, it would be possible to define another composition process. But there is no need to do the composition twice. The differentiation between vulnerabilities and unwanted incidents is probably more helpful for the composition than just having risks. For that reason, composition should be done only at threat analysis level.

A *threat composition diagram* with assets and consequence estimations can be used as the base to immediately identify and evaluate the risks without further need for component based composition. Hence, it is possible to create a conventional CORAS risk diagram for the entire system – without worrying about individual components anymore.

Just like in a conventional threat diagram, in a *threat composition diagram* each consequence relation *Q* leading from an unwanted incident *E* to an asset *T* is a risk *R*. The probability value of that unwanted incident *E* and the consequence value of *Q* are the parameters for the risk function, which is used to calculate the risk value for *R*. The risk value is necessary for applying the risk evaluation criteria. Risk functions, risk values and evaluation criteria are defined by the risk analysts in step 4 of the conventional CORAS method.

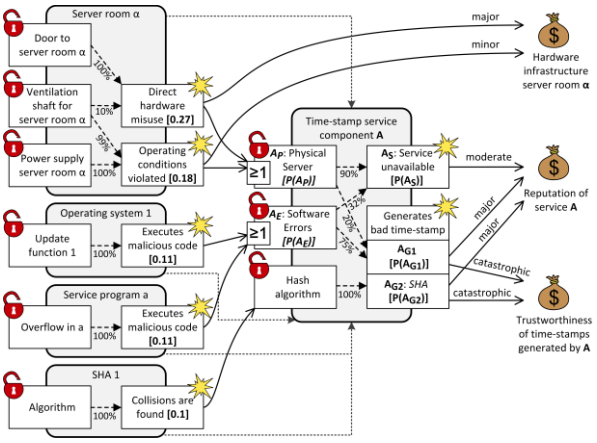

Figure 10. Threat composition diagram with assets and consequences

Typically, risks should be identified at a certain level of abstraction. For example, in the *threat composition diagram* shown in Figure 10, it would be possible to identify the risks at the high level of time-stamp component *A* or it would be possible to identify them at the low level of the base components. Only the unwanted incidents of the risk interfaces representing the components at the chosen level of abstraction are translated to risks for all possible consequences.

A consequence relation of some unwanted incident *E* can be indirect: If there is a path in the *directed graph of consequences* leading from *E* to some unwanted incident *W* having the consequence *Q* for asset *T*, then incident *E* can have the consequence *Q* for *T*, too. Hence, a risk can be identified for (*E*, *W*, *Q*, *T*) and the risk value can be calculated using the product of the absolute probability value for *E* and the relative likelihood for the path between *E* and *W* (i.e. $P(E) * P(W|E)$) as the first parameter and the consequence value of *Q* as the second parameter for the risk function.

For example, the unwanted incident "executes malicious code" of the "service program *a*" component in the example *threat composition diagram* shown in Figure 10 does not have direct consequences for any identified asset. But there are paths in the *directed graph of consequences* indicating that the incident can indirectly affect assets: There is one moderate consequence that the "executes malicious code" incident will have if it triggers the "service unavailable" incident and there are two consequences (one major, the other catastrophic) that it will have if it triggers the "generates bad time-stamp" incident.

Each of these indirect consequence relations leading from an incident to an asset is identified as an individual risk.

TABLE I. RISK FUNCTION FOR BASE INCIDENTS

		Consequences			
		minor	***moderate***	***major***	***catastrophic***
Likelihood	***<0.03***	very low	very low	low	medium
	[0.03-0.06[	very low	low	medium	high
	[0.06-0.16[	low	medium	high	very high
	≥0.16	medium	high	very high	very high

A common risk function defined for all base components in step 4 of the conventional CORAS risk analysis process is given in TABLE I. While the consequence value for a risk can just be read from the graph, the likelihood value for a risk has to be calculated along the path in the directed graph of consequences. The probability that the "executes malicious code" incident of component "service program *a*" occurs is 0.11, but only 32% of these incidents lead to the "service unavailable" incident. Therefore the probability for the risk "service program *a* executes malicious code" ("service unavailable") is 0.0352. Having a "moderate" consequence, this is a "low" risk.

Identifying and determining the risks in that way, it is possible to construct a flat conventional CORAS risk diagram using a *threat composition diagram* as input. It probably makes sense to summarize all the risks derived from the same unwanted incident in a compact structure as shown in Figure 11.

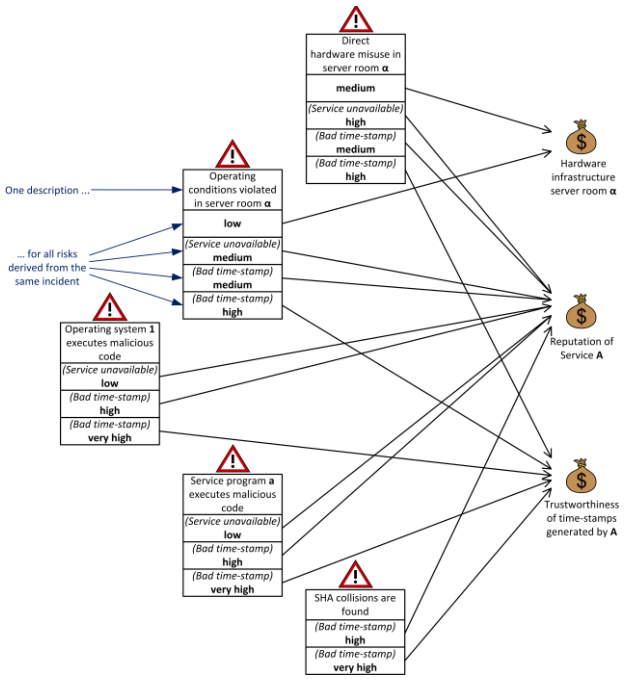

Figure 11. Risk diagram for a single time-stamp service

If only the unwanted incidents of risk interfaces for higher level components should be analyzed for identifying risks, caution is required if some base component incidents have consequences for assets that have not been identified for the higher level components. In the example shown in Figure 10, there are no consequences from the unwanted incidents of the "time-stamp service *A*" component for the "hardware infrastructure" asset. But these unwanted incidents can be triggered by the "direct hardware misuse" incident or by the "operating conditions violated" incident of the *risk interface* for the "server room α" component, which both have consequences for the "hardware infrastructure" asset. If these consequences and assets do not matter in the higher level context they may be ignored. Otherwise, the risk analysis for the higher level components was probably not complete and must therefore be repeated taking more assets into consideration.

A. Comparing the risks of components and architectures

Though it is possible to get completely rid of all the component and composition information when deriving risks from a *threat composition diagram*, it might also offer some benefits to create a diagram that keeps some information about the components. The idea is to make components or complex combinations of components comparable in terms of risks. Identifying the most critical components allows focusing treatment efforts. Typically, for a complex system, there is not only one single possible configuration. The system could probably be build using another combination of components or using completely other base components, too. It should be possible to choose the architecture with the fewest risks. Therefore, the *risk comparison diagram* is introduced here.

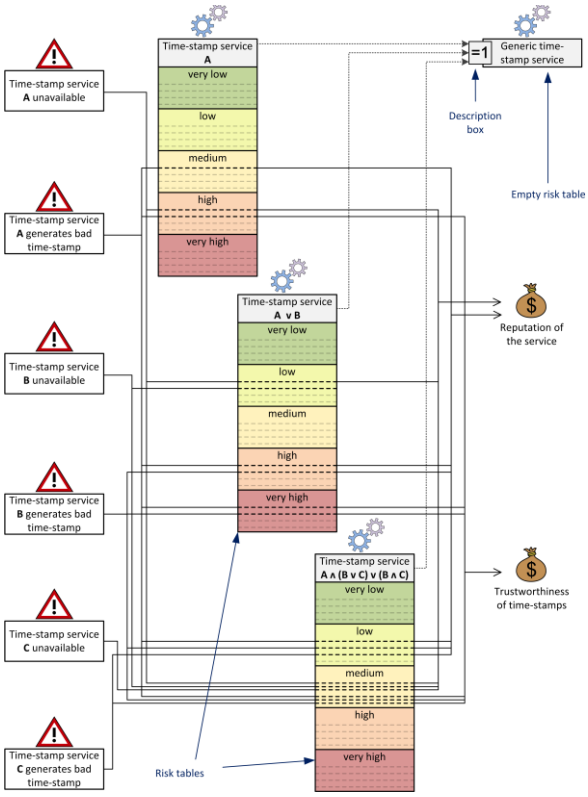

Figure 12. Risk comparison diagram

In a *risk comparison diagram*, each component is modeled as a *risk table*. Each *risk table* has a row for the component name and rows for all the risk values that have been defined during the risk analysis for that component. Relations between the components can be modeled between the *risk tables* with arrows having dashed lines and description boxes. A *risk comparison diagram* contains the risks and assets that can be identified for all involved components. In contrast to a risk diagram, the risk value is not written down for each risk. Instead, the consequence relations from risk *R* to asset *T* are made

through the *risk table* representing the component the risk was identified for. More precisely, the relations have to pass through the row representing the risk value of the risk *R*. That way, all risks of a component and their values are summarized in a *risk table* at a glance. Figure 12 shows a *risk comparison diagram* for three alternative time-stamp service designs.

For a complex system, there are many difficult design decisions, e.g. which individual components should be used and how much redundancy is optimal so that the remaining risks are acceptable. *Risk comparison diagrams* have proven to be a valuable tool for making high level decisions for the *S-Network* and they are helpful to communicate the decisions graphically.

V. Conclusion, Related and Further Work

With the extension presented here, the CORAS method becomes practicable for the risk analysis of large scale systems consisting of many different components like the *S-Network*. Modeling the relations between risk analyses artifacts generated for individual components, the probability values of unwanted incidents for the complex system can be calculated.

The *directed graph of consequences* in *threat composition diagrams* is similar to fault trees. It contains gates, which can express relations that conventional CORAS diagrams cannot model well. But in contrast to a fault tree, the *directed graph of consequences* does not have to be a tree: there can be multiple top level incidents. A single *directed graph of consequences* can represent multiple fault trees. Nodes in the *directed graph of consequences* modeling incidents can have relations leading to more than a single consequence incident. Therefore, dependencies can be modeled directly as common trigger nodes. In a fault tree, a fault triggering *n* other faults must be represented by *n* nodes having the same name but no graphical connection – which is less intuitive. Even more important, using the *directed graph of consequences*, bouncing analysis becomes feasible, going top-down and bottom-up with the same model. Using FTA, bouncing analysis is only possible in combination with other risk analysis methods like FMCA, which work on other models than fault trees. Transitions between different methods can cause problems and might be too difficult.

Containing the vulnerabilities, the external threats, the consequences and the assets, the *directed graph of consequences* offers the analyst more useful information than a fault tree. As a part of the *threat composition diagram*, the *directed graph of consequences* is always integrated in a model for the combination and interaction of the components themselves – represented by their *threat interfaces*. Having such a complete picture can help the analyst to identify all relevant risks.

However, diagrams can also get large and complex. High level CORAS is suggested to hide details in conventional CORAS diagrams [10]. A similar approach could be used for hiding parts in the new diagram types suggested here. A Software tool like the *CORAS tool* for the conventional CORAS method (http://coras.sourceforge.net/coras_tool.html) can help the analysts to deal with complex diagrams. Despite modeling and visualizing, a software tool could also support the computation of probability values in the *directed graph of consequences*.

Conventional risk diagrams can be created directly from the *threat composition diagram*. The least risky components and designs can be chosen using the *risk comparison diagram*.

Further research could try to add some information about the life cycle of unwanted incidents to the extended CORAS method. For how long does an unwanted incident last? Is the unwanted incident detected? Will the unwanted incident be repaired within a certain time-period once it was detected? Such information is essential to calculate more precise probability values. Established in other analysis methods like dynamic FTA, these aspects should be captured by CORAS, too.

References

[1] Zigmund Bluvband, Rafi Polak, Pavel Grabov: Bouncing Failure Analysis (BFA): The Unified FTA-FMEA Methodology, ALD Tel-Aviv 2005, http://www.aldservice.com/en/ articles/bouncing-failure-analysis-bfa-the-unified-fta-fmea-method.html (2012-04-15)

[2] Gyrd Brændeland, Heidi E. I. Dahl, Iselin Engan, Ketil Stølen: Using dependent CORAs diagams to analyse mutual dependency, LNCS 5141, Second International Workshop on Critial Information Infrastructures Security (CRITIS'07) pp. 135-148, Springer 2008

[3] Heidi E. I. Dahl, Ida Hogganvik, Ketil Stølen: Structured semantics for the CORAS security risk modelling language, 2nd International Workshop on Interoperability solutions on Trust, Security, Policies and QoS for Enhanced Enterprise Systems (IS-TSPQ'07). Report B-2007-3 pp. 72-92, University of Helsinki 2007

[4] Department of Defense: Procedings for Performing a Failure Mode, Effects and Criticality Analysis, MIL-STD-1629, Washington 1949/1980, http://www.fmea-fmeca.com/milstd1629.pdf (2012-04-15)

[5] Joanne Bechta Dugan, Kevin J. Sullivan, David Coppit: Developing a low-cost high-quality software tool for dynamic fault-tree analysis, Transactions on Reliability 2000-03 pp. 49-59, IEEE 2000, ISSN: 0018-9529, Digital Object Identifier: 10.1109/24.855536

[6] Clifton A. Ericson II: Fault Tree Analysis - A History, in Proceedings of the 17th International System Safety Conference, System Safety Society, Unionville 1999, http://www.fault-tree.net/papers/ericson-fta-history.pdf (2012-04-15)

[7] Rohit Gulati, Joanne Bechta Dugan: A Modular Approach for Analyzing Static and Dynamic Fault Trees, Proceedings of the 1997 Reliability and Maintainability Symposium in Philadelphia, PA pp. 57-63, IEEE 1997, Print ISBN: 0-7803-3783-2

[8] Ida Hogganvik, Ketil Stølen: A Graphical Approach to Risk Identification, Motivated by Empirical Investigations, 9th International Conference on Model Driven Engineering Languages and Systems 2006, LNCS 4199 pp. 574-588, Springer Berlin Heidelberg 2006, DOI: 10.1007/11880240_40

[9] Andrei Kolmogorov: Grundbegriffe der Wahrscheinlichkeitsrechnung, Springer Verlag Berlin 1933,

[10] Mass Soldal Lund, Bjørnar Solhaug, Ketil Stølen: Model-Driven Risk Analysis, The CORAS Approach, Springer Verlag Berlin Heidelberg 2011, ISBN: 978-3-642-12322-1

[11] Antoine Rauzy: New algorithms for fault trees analysis, Reliability Engineering and System Safety 40 (1993) pp. 203-211, Elsevier Science Publishers 1993

[12] Michael Stamatelatos, Joanne Dugan, Joseph Fragola, Joseph Minarick, Jan Railsback: Fault Tree Handbook with Aerospace Applications, NASA, Washington 2002, http://www.hq.nasa.gov/office/codeq/doctree/fthb.pdf (2012-04-15)

[13] W. E. Vesely, F. F. Goldberg, N. H. Roberts, D. F. Haasl: Fault Tree Handbook, U.S. Nuclear Regulatory Commission, Washington 1981, http://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0492/sr0492.pdf (2012-04-15)

[14] Johannes Viehmann: The Theory of Creating Trust with a Set of Mistrust-Parties, proceedings of PST 2012 in Paris, IEEE 2012

[15] H. A. Watson: Launch Control Safety Study, Section VII, Vol 1, Bell Laboratories, Murray Hill 1961