Safety, Reliability, Risk and Life-Cycle Performance of Structures & Infrastructures – Deodatis, Ellingwood & Frangopol (Eds)
© 2013 Taylor & Francis Group, London, ISBN 978-1-138-00086-5

The CORAS Language – why it is designed the way it is

Bjørnar Solhaug
SINTEF ICT, Norway

Ketil Stølen
SINTEF ICT, Norway
Department of Informatics, University of Oslo, Norway

1 INTRODUCTION

CORAS[1] [6] is an approach to risk analysis based on the ISO 31000 international standard on risk management [4]. The approach is model-driven in the sense that graphical models are actively used throughout the whole risk analysis process to support the various analysis tasks and activities, and to document the results. It is defensive, which means that the risk analysis is concerned with protecting existing assets, rather than balancing potential gain against risk of investment loss (as, for example, within gambling or stock trading). It is asset-driven in the sense that the assets to be defended and protected are identified during the very initial phases of the process; all subsequent tasks, such as risk identification and risk treatment, are driven by these assets to ensure that the analysis focuses on what the risk analysis should help to defend.

CORAS is a self-contained approach to risk analysis in the sense that it comes with all guidelines, techniques and tool-support that are needed throughout the whole process. In particular, CORAS consists of the three tightly interwoven artifacts of a language, a tool and a method. The tool supports the CORAS language, and hence all steps of the method. The method is divided into eight steps with clearly defined objectives. Each step is decomposed into specific analysis tasks supported by practical guidelines for how to conduct the tasks in practice using the tool and the language.

Clearly, when developing any such approach to risk analysis, it is necessary to make a number of design choices. The choices are obviously determined by the kind of approach we aim for, and what we wish the users of the approach should be able to accomplish. At the same time, different objectives may pull in different directions and therefore need to be balanced. In this paper we present the most important features of the CORAS language and motivate some of the major design choices we did. More precisely, in Section 2 we give an overview of the language and the different kinds of diagrams that are supported. In Section 3 we give examples of two CORAS diagrams and explain their main features and uses. In Section 4 we motivate some of the major design decision concerning issues such as comprehensibility, expressiveness, formality, scalability and change management. In Section 5 we focus on the analysis and documentation of likelihood, which is a core part of any risk analysis. Finally, in Section 6, we conclude.

[1] See the web site for resources on CORAS, including the CORAS tool for free download: http://coras. sourceforge.net/

2 OVERVIEW OF THE CORAS LANGUAGE

The CORAS language is designed to facilitate and support the reasoning about and the documentation of different aspects of risks. These are aspects that are needed in order to identify and understand risks, the risk sources, and the adequate means for mitigating unacceptable risk. The language therefore has syntactical constructs for capturing precisely such aspects. As much as possible, the language has been designed to express concepts and terms as they are commonly used within existing risk management frameworks and international standards [4, 5]. To this end, the CORAS approach is built on a conceptual framework of precisely defined terms, and the CORAS language is in turn designed to make expressions in these terms and with a semantics that captures their definitions.

The language offers different kinds of diagrams that each supports specific tasks and phases of the risk analysis process. The *basic CORAS language* consists of the five kinds of diagrams that are most typical in CORAS risk analyses, namely *asset diagrams*, *threat diagrams*, *risk diagrams*, *treatment diagrams*, and *treatment overview diagrams*. The diagrams are usually used in this order, and one kind of diagram serves as input to the step that makes use of the next kind of diagram.

Asset diagrams are employed during context establishment as part of the documentation of the target of analysis. As the name indicates, an asset diagram is mainly used for the purpose of defining the assets of relevance for the analysis. The diagram specifies the party that assigns value to the assets in question, as well as the relations between the assets.

Threat diagrams are used during risk identification and risk estimation. A threat diagram expresses

how threats exploit vulnerabilities to initiate threat scenarios that lead to unwanted incidents that harm the identified assets. Threat diagrams also support the specification of and reasoning about likelihoods and consequences.

Risk diagrams provide an overview of the identified risks, where each risk is assigned a risk level as derived from likelihood and consequence estimates. Essentially, a risk diagram gives a compressed overview of the findings documented in the threat diagrams, making each risk explicit and showing only the threats that initiate them and the assets that are harmed. Risk diagrams give a basis for deciding which risks should be considered for risk treatment.

Treatment diagrams are a kind of extended threat diagrams that are used to identify and document mitigation options for unacceptable risks. Treatment diagrams come with a specific risk treatment construct to document how treatments can be applied to threats, vulnerabilities, threat scenarios or unwanted incidents to reduce the likelihood and/or consequence or risks.

A treatment overview diagram is similar to a risk diagram. It gives an overview of the risks and the identified treatments to highlight which risks are mitigated by which treatments.

In addition to these five kinds of diagrams of basic CORAS, further modeling and analysis support is provided by three extensions, namely *high-level CORAS*, *dependent CORAS* and *legal CORAS*. High-level CORAS is for hierarchical modeling at different levels of abstraction, and is a means for providing a comprehensible overview of large risk models. Dependent CORAS is designed to support the explicit documentation of analysis assumptions and analysis dependencies, and to support modular reasoning. Legal CORAS supports the identification and documentation of legal aspects that may affect risks, as well as the level of impact of legal aspects on risk.

3 SMALL EXAMPLE

In this section we illustrate the CORAS language with two diagrams from an example of an information security risk analysis. The diagrams present some of the most important constructs of the language, they illustrate the graphical appearance of CORAS diagrams, and they show how the language supports key aspects of the documentation of and reasoning about risk. The assumed target of analysis in the example is an online store that customers can access via the Internet by using a web application. The client of the risk analysis is the service provider, i.e. the owner of the online store.

Figure 1 depicts an asset diagram. The party in this diagram is the *service provider*. Three of the assets are so-called *direct assets*. *Compliance* denotes compliance with data protection laws and regulations, (information) *security* is the preservation of confidentiality, integrity and availability of information, whereas *service provisioning* is the availability and quality of the online store services. The fourth asset,

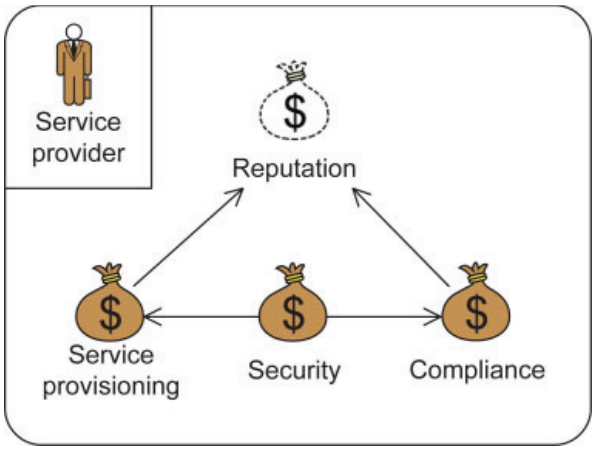

Figure 1. Example of a CORAS asset diagram.

namely the company *reputation*, is a so-called *indirect asset*. The latter is an asset that, with respect to the target of analysis, is harmed only via harm to other assets. The risk identification is conducted with respect to the direct assets only, so the use of this distinction between assets is a means to keep the risk analysis focused. Once risks have been identified for the direct assets, the possible further consequences with respect to the indirect assets can be analyzed. The implications of risks to one asset for other assets are captured by the so-called *harm relations*.

When conducting risk identification using CORAS threat diagrams, we start with the identified assets. In the example shown in Figure 2, these are depicted to the right. In threat diagrams risks are identified by the identification of threats, vulnerabilities, threat scenarios and unwanted incidents. In the example there are two threats, namely *hacker* and *employee*. Both of these threats are human, but the former is a deliberate threat and the latter is accidental. Additionally the CORAS language offers a separate construct for modeling non-human threats (such as virus, fire, flood, etc.), but this is not exemplified here. A threat can initiate threat scenarios and unwanted incidents. In the example diagram *hacker* initiates the threat scenario *malcode introduced by hacker via web application*. The diagram also documents a vulnerability that this threat may exploit, namely the *use of web application*. A threat scenario can lead to other threat scenarios and to unwanted incidents. For example, the threat scenario *disclosure of customer data* can lead to the unwanted incident *leakage of personally identifiable information*. The unwanted incidents constitute the risks, and each incident is documented with its likelihood, the assets it harms, and its consequence for each asset. For example, *leakage of personally identifiable information* harms both *compliance* and *security*; it therefore constitutes two risks, one for each asset it harms.

The example also shows the estimation and documentation of likelihoods and consequences, both of which can be qualitative or quantitative. In CORAS, the likelihoods and consequences can also be specified as exact values or using intervals. In the example there is a mix of exact frequency values and intervals; 20 : 1*y*

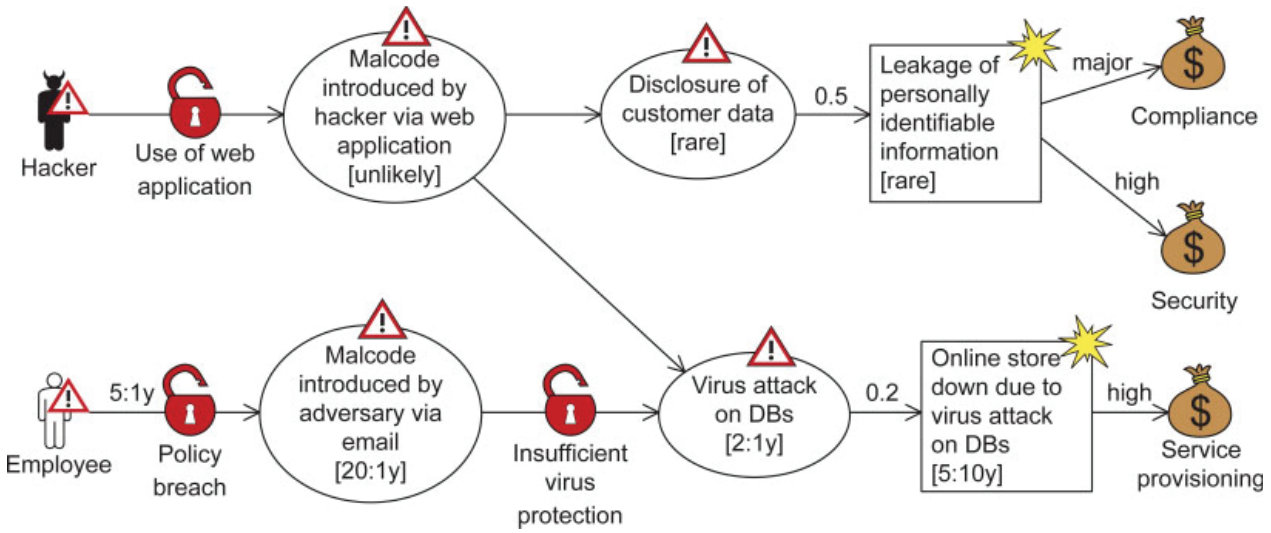

Figure 2. Example of a CORAS threat diagram.

denotes the frequency of 20 times per year, whereas *rare* and *unlikely* are defined as the intervals $[0, 2\rangle$ and $[2, 5\rangle$ per year, respectively.

The frequency on the relation from a threat is the estimate of the threat to initiate the following scenario or incident. For example, *employee* initiates *malcode introduced by adversary via email* five times per year.

The documented likelihoods serve as a basis for calculating unknown likelihoods or for checking consistency of existing estimates in the diagrams. In any case, when reasoning about likelihoods we need to determine whether a diagram is complete. For example, if the diagram in Figure 2 is complete, *virus attack on DBs* is the only scenario that can lead to the incident *online store down due to virus attack on DBs*. Due to the conditional likelihood of 0.2, we can calculate the likelihood of the incident to $(2 : 1y) \cdot 0.2 = 0.4 : 1y = 4 : 10y$. This is inconsistent with the likelihood $5 : 10y$ already assigned to this incident.

As another example, consider the likelihood $20 : 1y$ of the threat scenario *malcode introduced by adversary via email*. According to the diagram, *employee* initiates this scenario five times a year. Therefore, if the diagram is complete the estimates are inconsistent. However, if there may be other occurrences of infected emails than policy breaches and sloppy employees, the frequency of the scenario can be higher. Hence, if the diagram is incomplete the estimates are consistent.

Systematically detecting inconsistencies using the CORAS calculus and guidelines is an important part of validating or correcting the risk analysis results. The calculus can be applied to single relations in the diagrams, but also to calculate or consistency check combined scenarios. For example, the likelihood of *virus attack on DBs* can be checked by combining the likelihoods of the two preceding threat scenarios.

4 MAJOR DESIGN DECISIONS

Creating a modeling language involves making many design decisions. Such decisions are often not well-documented and can be difficult to trace. In the following we shed light on why the CORAS language has become as it is.

4.1 *Comprehensibility*

The CORAS method makes use of structured brainstorming as a technique for risk identification and estimation. During such sessions the results are documented on-the-fly by risk modeling. To capture as many viewpoints as possible and to gather complementary knowledge and experience, the brainstorming involves personnel with different roles and different experiences with respect to the target of analysis. In the setting of a CORAS risk analysis, the CORAS language is a key facilitator for communication and to ensure a common understanding.

For this purpose the CORAS language has been developed and designed with comprehensibility as the guiding principle. Some of the important characteristics of the language in this respect are the following.

First, the language is based on established risk terminology. The CORAS approach includes a conceptual framework of relevant terms that are closely based on common practices and international standards. The CORAS language is built on top of these concepts with separate language constructs for the different terms. When involving personnel in a CORAS risk analysis, the underlying concepts are carefully explained with systematic associations to the graphical constructs. The explicit traceability to the underlying risk concepts is intended to further increase the comprehensibility of the CORAS language.

Second, the graphical icons are designed to match intuition [2, 3]. During the development of the language, many different design options were proposed and evaluated empirically in trials and experiments in relevant user groups. These empirical studies investigated the extent to which the intuitive interpretation of the meaning of the icons matched their intended meaning, and also which kinds of shapes, sizes and colors

most efficiently facilitated the intended understanding and communication.

Third, the language is diagrammatic with a simple syntax and few constructs. As an alternative to a diagrammatic language, risks can be documented, for example, using tables or other more prose-based formats. However, prose and tables are less suitable for capturing relations between different threats, vulnerabilities, unwanted incidents, etc. in the simple and straightforward fashion that is required during brainstorming. Moreover, by keeping the CORAS syntax simple, people involved in a risk analysis are more likely to grasp which risks are expressed and due to which threats and vulnerabilities, without having to do tedious parsing or carefully check what the different constructs denote. There is of course no limit on the size of the diagrams, but in our pragmatic guidelines for how to best model using CORAS we recommend making diagrams that fit on a standard sheet of paper or is readable on a canvas.

Fourth, the diagrams hide subtleties that only the analysts need to consider. For example, most relations are expressed with the same kind of arrow although they in reality may be different both syntactically and semantically. The differences are, however, mostly relevant for the analysts when conducting more thorough analyses outside the brainstorming sessions and workshops. Furthermore, the soundness of the CORAS calculus, that applies to the diagrams, depends on an underlying formality that brainstorming participants and other stakeholders do not need to relate to or understand while making their contribution to the risk identification and estimation.

4.2 *Expressiveness*

A model of the reality is always an abstraction. That a model is an abstraction means that there are aspects and features of reality that are not present in the model. What parts or ingredients of reality you can express in a model depend to a large extent on the modeling language you use. Some modeling languages are more expressive than others in the sense that they can model reality at a finer level of granularity. Some modeling languages are specialized towards certain domains, while others are more general purpose. For example, predicate logic is a modeling language that can be used within almost any area. Modeling languages designed for very specific domains are often referred to as domain specific, and the CORAS language is clearly a domain specific language.

To select which features of reality that a modeling language should support is a major decision for a language designer. The CORAS language has been designed to support risk modeling based on brain-storming conducted during a risk analysis. However, this does not mean that the CORAS language is suited to model anything of relevance for a risk analysis. To make the CORAS language more expressive is easy, but more expressiveness is costly in the sense that it tends to increase complexity and reduce comprehensibility. The result might be a CORAS language that is no longer suited as a medium for communication during brainstorming sessions; i.e., a language no longer usable for its original purpose.

One highly relevant aspect of risk analysis that the CORAS language is not intended to model is the target of analysis. More precisely, the CORAS language is a language for describing the risky behavior of relevance for the target of analysis; it is not a language for describing the target's intended behavior and design. There are several reasons for this. One reason is the already mentioned need to maintain comprehensibility and simplicity. Another reason is that there are already many languages well-suited for this purpose, such as UML [8] and BPMN [9]. A third reason is that some clients have their own target models expressed in languages they are familiar with, and it might be counterproductive to force them to discuss their systems and processes in notations that they are not used to.

Although CORAS has been used within many different fields it is nevertheless the case that CORAS has mainly been designed for defensive risk analysis with particular focus on security. By defensive we mean risk analysis where the main purpose is to defend the assets that you are already in possession of as is normally the case within security risk analysis. The focus on security is mainly reflected in the language through reserved terms like "threat", "threat scenario" and "vulnerability". The focus on security is rather shallow. Since it is up the user to define the assets to be protected, the CORAS language may for example just as well be used within the safety field, although some safety experts may find the above mentioned terms too security oriented. The focus on defensive risk analysis is more fundamental.

4.3 *Formality*

Within modern science the interpretation of phrases like "formal", "formal model" and "formal language" varies from one area or specialized field of interest to another. For example, some use "formal" in the meaning of "mathematical". But this conflicts with traditions within mathematics and logic. Elementary geometry was formalized by Euclid in the form of an axiomatic system, but geometry as a mathematical field is much older than Euclidean geometry. Moreover, the axiomatic system of Euclid had several deficiencies and flaws some of which were discovered fairly recently. Nevertheless, the flaws of Euclidean geometry have not had any effect on the success of geometry in science and engineering. Hence, formal is not necessarily the same as mathematical, and a correct formalization is not a pre-condition for the success of mathematics.

In this paper we use the term "formal" in the following interpretation: a language is formal if there is an algorithm capable of deciding whether an expression in the syntax of the language is grammatically correct.

In this respect the CORAS language is formal, and this has many advantages:

- Formality makes it possible to provide a semantics allowing models to be understood independently of the analyst or those involved in making them.
- Formality facilitates the development of general rules and techniques for reasoning and mathematical analysis of models expressed in the language.

By a semantics for a language we mean a translation of the grammatically correct expressions of the language into a notation that is well-understood for the intended users of the semantics. The CORAS language comes with two kinds of semantics intended for two different purposes. The first kind is a schematic translation of any grammatically correct expression in the CORAS language into English prose [6]. This is the semantics intended for the everyday CORAS users. It may be referred to during a CORAS analysis to decide what some fragment of the CORAS language actually denotes, and it may be used by third parties reading the report from a CORAS analysis to determine the meaning of the CORAS diagrams in the report. The second kind of semantics is a schematic translation of any grammatically correct expression into mathematics [1]. This semantics is mainly intended for the developers of the CORAS method and tools, and provides the foundation for the general rules and techniques for reasoning and mathematical analysis. For example, the soundness proof of the CORAS calculus is based on this semantics.

When we claim that CORAS is formal we have basically interpreted the text decorating the icons for threats, vulnerabilities, threat scenarios etc. as strings of text that are translated one-to-one by the above mentioned semantic mappings. This is of course a considerable simplification. This text is normally expressed in some natural language. In this paper we have used English, but it could equally well be Chinese or Arabic. This part of the CORAS language is highly informal. It could of course be formalized, for example by requiring the use of predicate logic instead. This would facilitate a much more advanced mathematical analysis, but we would at the same time make the language unsuitable for its intended purpose since very few of those involved in a risk analysis are trained logicians.

4.4 *Scalability*

In risk modeling and risk analysis, scalability is the ability of the modeling and analysis techniques to efficiently cope with large-scale systems. The CORAS language includes two kinds of diagrams to deal with the scalability problem, namely high-level CORAS and dependent CORAS.

High-level CORAS is essentially a means for hierarchical risk modeling that supports both detailing and abstraction. Detailing means to decompose an element of a CORAS diagram in order to analyze this element at a lower level of abstraction. For example, the threat scenario *malcode introduced by hacker via web application* depicted in Figure 2 does not document exactly how a hacker may go by to do this attack, and which vulnerabilities of the web application that may be exploited. Using high-level CORAS, this threat scenario can be decomposed in a separate diagram where a more detailed analysis and modeling is conducted. This is illustrated in Figure 3 where different kinds of attacks are specified, and how they can lead to other scenarios. If further detailing is needed, all of these scenarios can be further decomposed in new diagrams. Note also that high-level CORAS uses gates on the outline of the constructs to keep track of the decomposition. In the example i_2 is a so-called *in-gate* and o_1 is an *out-gate*.

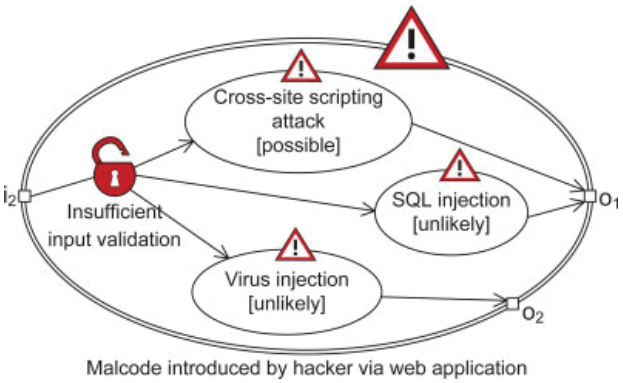

Figure 3. Decomposed threat scenario using high-level CORAS.

The CORAS diagram elements that can be decomposed using high-level CORAS are threat scenarios, unwanted incidents, risks and treatments. In the opposite direction, high-level CORAS can be used to combine several diagrams at a higher level of abstraction to provide an overview of the broader risk picture while maintaining traceability to the low-level details.

Dependent CORAS is an extension of the CORAS language to facilitate the explicit modeling and reasoning about risk analysis assumptions. The diagrams are divided into two, namely the target T and the assumptions A, both of which are threat diagrams or fragments thereof. Basically, T is a threat diagram regarding the chosen target of analysis, and A are threat diagrams on which T depends. For large systems or mutually dependent systems (such as systems of systems), dependent CORAS allows the target to be split into several subsystems that are analyzed separately in a modular way. Because the risks in one sub-system may depend on the risks in another, these dependencies must be taken into account, and this is precisely what dependent CORAS is designed for. In particular, what makes the assumptions with respect to one sub-system may be part of the target in another, and vice versa. Dependent CORAS comes with modeling support to capture this, as well as a calculus with rules for how to reason about the assumptions and deduce the valid risk picture for the different sub-systems.

4.5 *Change*

Traditional approaches to risk analysis typically focus on a particular configuration of the target at a

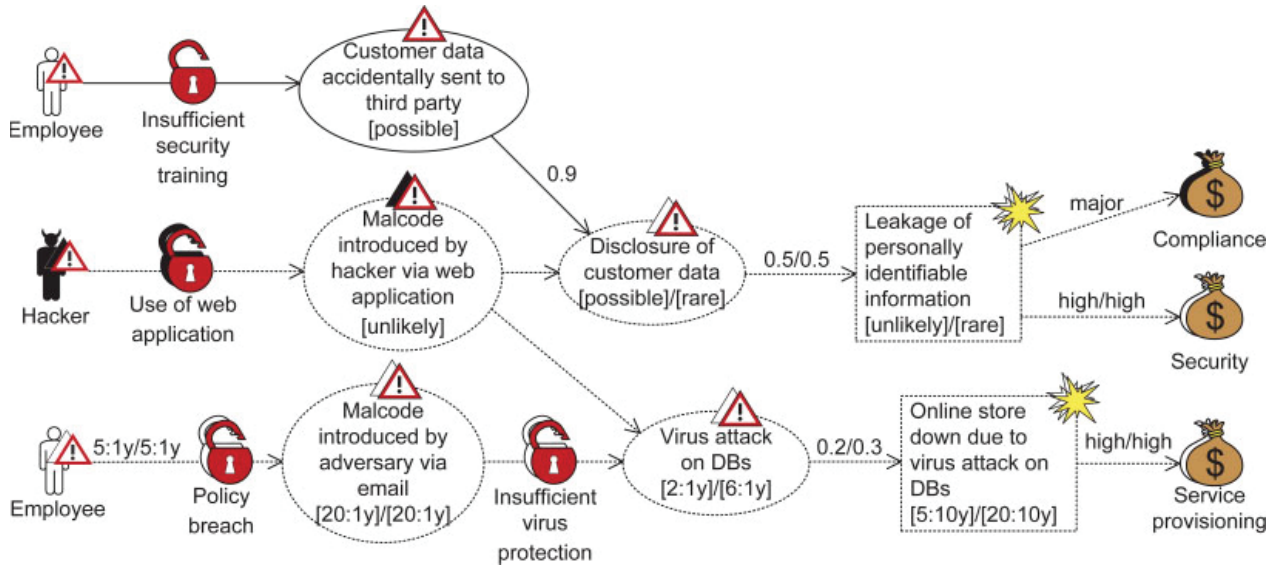

Figure 4. CORAS diagram with change.

particular point in time, and is valid under the assumptions made in the analysis. However, the target of analysis, its environment and the assumptions we make will change and evolve over time. Such changes may render previous risk analyses invalid and require them to be conducted again from scratch.

In order to systematically handle change, CORAS offers support for modeling and analyzing evolving risks [7, 12]. The objective is to maintain the validity of the risk model and the risk analysis results while systems change. Moreover, when risks are changing and evolving they should be analyzed and understood as such. In other words, the analysis of evolving risks should be supported by techniques for modeling and reasoning about risk changes. Such techniques will allow planning and proactive decisions regarding potential planned or foreseen changes.

Specifically, the CORAS language allows the specification of risk elements that become obsolete after change, risk elements that emerge, and risk elements that are modified. For this purpose, all language constructs can be assigned one of the modes *before*, *after* and *before-after*, respectively.

As illustrated in Figure 4, each such mode has its own graphical design to ensure that the changes are explicitly visualized in the diagrams. In this example we assume that the web application has just been launched, and that previously customers could place orders by email only after browsing the assortment on the company web site. Hence, the threat scenario *malcode introduced by hacker via web application* emerges and is assigned the mode *after*. Elements of this mode are depicted by two layered icons with black background. Note that also the asset *compliance* has mode *after*, which shows that the issue of data protection was considered only after the launch of the web application.

The threat scenario *customer data accidentally sent to third party* is an example of the mode *before*. These elements are depicted by one layered icons as in standard CORAS. In the example we assume that security training and routines have improved to the extent that this source of risk is insignificant and can be ignored.

The threat scenario *malcode introduced by adversary via email* is an example of the mode *before-after*. These elements are depicted by two layered icons with white background. Moreover, *before-after* elements with likelihoods are assign a pair of likelihoods. The former specifies the likelihood before change, and the latter the likelihood after change. For example, the likelihood of the unwanted incident *online store down due to virus attack on DBs* increases from $5 : 10y$ to $20 : 10y$.

Although not illustrated here, the language also supports relating the CORAS diagram elements to elements of the target model. The specification of these relations is referred to as the *trace model*, as it facilitates systematic traceability of changes from the target model to the risk model. This is an important feature, since it allows analyses to be updated by considering only the CORAS diagrams that may be affected by the changes. Hence, the CORAS diagrams with change, as exemplified in Figure 4, are not built from scratch; instead, CORAS diagrams from previous analyses serve as input and are systematically updated and revised in order to analyze the risk changes.

A challenge with introducing change as a dimension of its own in the diagrams is that it may be at the cost of comprehensibility and intuitive understanding. The extension was therefore as modest as possible, both at the level of syntax and semantics.

5 DOCUMENTING AND ANALYZING LIKELIHOOD

An essential aspect of any risk modeling language is the representation of likelihood. In CORAS it is up to the analyst to define the likelihood scale. You may use purely qualitative values or you may work with quantitative values. In the latter case, you may freely choose

between frequencies and probabilities. Furthermore, you may employ a continuous scale or a discrete scale; you may also operate with intervals. There are two main reasons for this flexibility:

1. The likelihood data available varies considerably from client to client and between different targets of analysis.
2. Large clients often have predefined scales that any risk analysis must be conducted according to.

Hence, without this flexibility we would basically put ourselves out of business.

Taking this as given there are of course nevertheless many ways to support likelihood documentation as well as analysis and reasoning based upon this documentation. In the following we will motivate the treatment of likelihood in the CORAS language.

5.1 *Aleatory versus epistemic uncertainty*

Uncertainty is often classified into two kinds [10]. On the one hand we may be uncertain about the future due to ignorance and lack of evidence. On the other hand uncertainty may be due to the inherent randomness of systems. The latter kind of uncertainty is commonly referred to as aleatory uncertainty and pertains to chance. Typical examples are the outcomes of the tossing of a coin, or the hands players of a game of poker receives. Aleatory uncertainty is the inherent randomness that cannot be removed from systems (without redesigning the systems). The former kind of uncertainty is commonly referred to as epistemic uncertainty and pertains to our knowledge about the system at hand. When making predictions about future behavior, the epistemic uncertainty is something we actively seek to reduce by gathering more information and evidence. In CORAS we do not distinguish between aleatory and epistemic uncertainty; the reason is quite simply that we do not think such a distinction is feasible in practice. In CORAS, epistemic uncertainty is normally captured only implicitly by the use of intervals. In this respect we are very much in line with the conclusions in [11]. Reducing epistemic uncertainty would then correspond to introducing a scale with finer likelihood intervals.

5.2 *State-full versus state-free*

A CORAS threat diagram is basically a set of finite paths starting from threats and ending in assets. Along these paths there are nodes in the form of threat scenarios and unwanted incidents, and relations between them. The relations may be decorated with vulnerabilities. This is about as simple as a threat model can become, and keeping things simple has been a major objective in the design of the CORAS language. When it comes to likelihoods our main emphasis has been to maintain this simplicity. Likelihood values may be assigned to relations and nodes already presents in the paths, but we have refrained from introducing more complex annotations, like the state-dependent annotations of a general Bayesian network. Of course this additional expressiveness could be nice and useful in certain situations, but our experience as risk analysts is that the data it requires is very seldom available in practice.

5.3 *Consistency versus inconsistency*

As already mentioned, likelihoods may freely be assigned to relations and nodes in a CORAS threat diagram. As a result, one may easily end up with inconsistencies. Based on the already mentioned emphasis on simplicity some might expect the likelihood annotations to be constrained in such a way that inconsistencies cannot occur. But this would reduce the value of the CORAS language because as a risk analyst you actually hunt for inconsistencies. That things appear nice and polished is a good indication that the relevant aspect is well understood, and that the analyst should look elsewhere for hidden risks. On the other hand, when the analysis team provides inconsistent estimates things are obviously not as they appear, and in such cases there may be hidden risks involved. In other words, inconsistencies may indicate the need for further investigation and a more thorough analysis.

5.4 *Crisp versus fuzzy*

Fuzzy logic provides a simple way to draw definite conclusions from vague, ambiguous or imprecise information, and allows partial membership in a set. It allows modeling complex systems using higher levels of abstraction originating from the analyst's knowledge and experience [14]. A fuzzy set is a class of objects with a continuum of grades of membership. Such a set is characterized by a membership function, which assigns to each object a grade of membership ranging between zero and one [15]. Using the fuzzy membership functions, a parameter in a model can be represented as a crisp number, a crisp interval, a fuzzy number or a fuzzy interval. In the fuzzy logic approach the algebraic operations are easy and straightforward, as argued and elaborated in [13]. The interval scales we use in CORAS is a special case of the fuzzy approach, where only the crisp intervals are used as membership functions. Whether and to what extent CORAS would benefit from fuzzy reasoning is an open question that we intend to study in further detail in the future.

6 CONCLUSIONS

In this paper we have given a brief overview of the CORAS language and motivated some of the major design decisions on which it builds. In particular, we have emphasized the importance of simplicity. A major challenge for a language designer is to find the right balance between expressiveness and comprehensibility. We have also briefly outlined how the CORAS language copes with difficult challenges like scalability and change, and why there is such a strong focus on assets in all kinds of CORAS diagrams. We have

also motivated the CORAS approach to capturing and reasoning about likelihoods. In particular, we have explained why the CORAS language does not distinguish between different kinds of uncertainty, why there are no states and why the possibility of expressing inconsistencies is a feature.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the Research Council of Norway via the DIAMONDS project (201579/S10), and from the European Union's Seventh Framework Programme (FP7/2007-2013) via the NESSoS network of excellence (256980) and the RASEN project (316853).

REFERENCES

[1] G. Brændeland, A. Refsdal, and K. Stølen. Modular analysis and modelling of risk scenarios with dependencies. *Journal of Systems and Software*, 83(10):1995–2013, 2010.

[2] I. Hogganvik and K. Stølen. A graphical approach to risk identification, motivated by empirical investigations. In *Model Driven Engineering Languages and Systems (MoDELS'06)*, volume 4199 of *LNCS*, pages 574–588. Springer, 2006.

[3] I. Hogganvik. *A Graphical Approach to Security Risk Analysis*. PhD thesis, University of Oslo, 2007.

[4] International Organization for Standardization. *ISO 31000 – Risk management – Principles and guidelines*, 2009.

[5] International Organization for Standardization. *ISO Guide 73 – Risk management – Vocabulary*, 2009.

[6] M. S. Lund, B. Solhaug, and K. Stølen. *Model-Driven Risk Analysis – The CORAS Approach*. Springer, 2011.

[7] M. S. Lund, B. Solhaug, and K. Stølen. Risk analysis of changing and evolving systems using CORAS. In *Foundations of Security Analysis and Design VI (FOSAD'VI)*, volume 6858 of *LNCS*, pages 231–274. Springer, 2011.

[8] Object Management Group. *OMG Unified Modeling Language (OMG UML), Superstructure. Version 2.2*, 2009. OMG Document: formal/2009-02-02.

[9] Object Management Group. *Business Process Model and Notation (BPMN). Version 2.0*, 2011. OMG Document: formal/2011-01-03.

[10] T. O'Hagan. Dicing with the unknown. *Significance*, 1(3):132–133, 2004.

[11] A. Omerovic and K. Stølen. A practical approach to uncertainty handling and estimate acquisition in model-based prediction of system quality. *International Journal on Advances in Systems and Measurements*, 4:55–70, 2011.

[12] B. Solhaug and F. Seehusen. Model-driven risk analysis of evolving critical infrastructures. *Journal of Ambient Intelligence and Humanized Computing*, 2013. To appear. Available electronically, DOI 10.1007/s12652-013-0179-6.

[13] P. V. Suresh, A. K. Babar, and V. Venkat Raj. Uncertainty in fault tree analysis: A fuzzy approach. *Fuzzy Sets and Systems*, 83(2):135–141, 1996.

[14] D. P. Weber. Fuzzy fault tree analysis. In *Proc. 3rd IEEE Conference on Fuzzy Systems, 1994. IEEE World Congress on Computational Intelligence.*, pages 1899–1904. IEEE, 1994.

[15] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1926.