Capturing Security Requirements through Misuse Cases

Guttorm Sindre
Dept of Computer and Info. Sci.
Norwegian Univ. of Sci. and Tech.
guttorm@idi.ntnu.no

Andreas L. Opdahl
Dept of Information Science
University of Bergen, Norway
andreas@ifi.uib.no

Abstract

Use cases have become popular for eliciting, communicating and documenting requirements. They support functional requirements well, but provide less support for working with extra-functional requirements, such as security requirements. With the advent of e- and m-commerce applications, such requirements are growing in importance. This paper discusses a conceptual extension of use cases, namely 'misuse cases', describing actions that should not be possible in a system. In particular, we discuss templates for their textual description.

Keywords: Use cases, extra-functional requirements, security.

1. Introduction

Use cases [1,2,3] have proven helpful for the elicitation of, communication about and documentation of requirements [4,5]. Earlier research indicates that software development projects where the analysis phase concentrates on use cases rather than textual requirements may be more successful in capturing the user needs [6,7]. But there are also problems with use case based approaches to requirements engineering. As stated in [8]: "A collection of use cases is not a suitable substitute for a requirements specification". Typical problems are over-simplified assumptions about the problem domain [9] and a tendency to go prematurely into design considerations, especially concerning the user interface [2,10]. Moreover, use cases are not equally suited for all requirements and thereby introduce bias in the elicitation process.

A use case typically describes some function that the system should be able to perform [3]. Hence, use cases are good for working with so-called functional requirements, but not necessarily with extra-functional ones, such as security [11,12]. These will seldom be stated directly by the stakeholders, who rather have *concerns* about what should *not* happen in the system [13,14]. Use cases, by their nature, concentrate on what the system *should* do, and have less to offer when describing the opposite. But system behavior that should be avoided is also behavior, which could potentially be investigated through use cases. This motivated the extension of use case concepts proposed in [11]:

- A *misuse case* is the inverse of a use case, i.e., a function that the system should not allow. Just as [1] defines a use case as a completed sequence of actions which gives increased value to the user, one could define a misuse case as a completed sequence of actions which results in loss for the organization or some specific stakeholder.
- A *mis-actor* is the inverse of an actor, i.e., an actor that one does not want the system to support, an actor who initiates misuse cases.

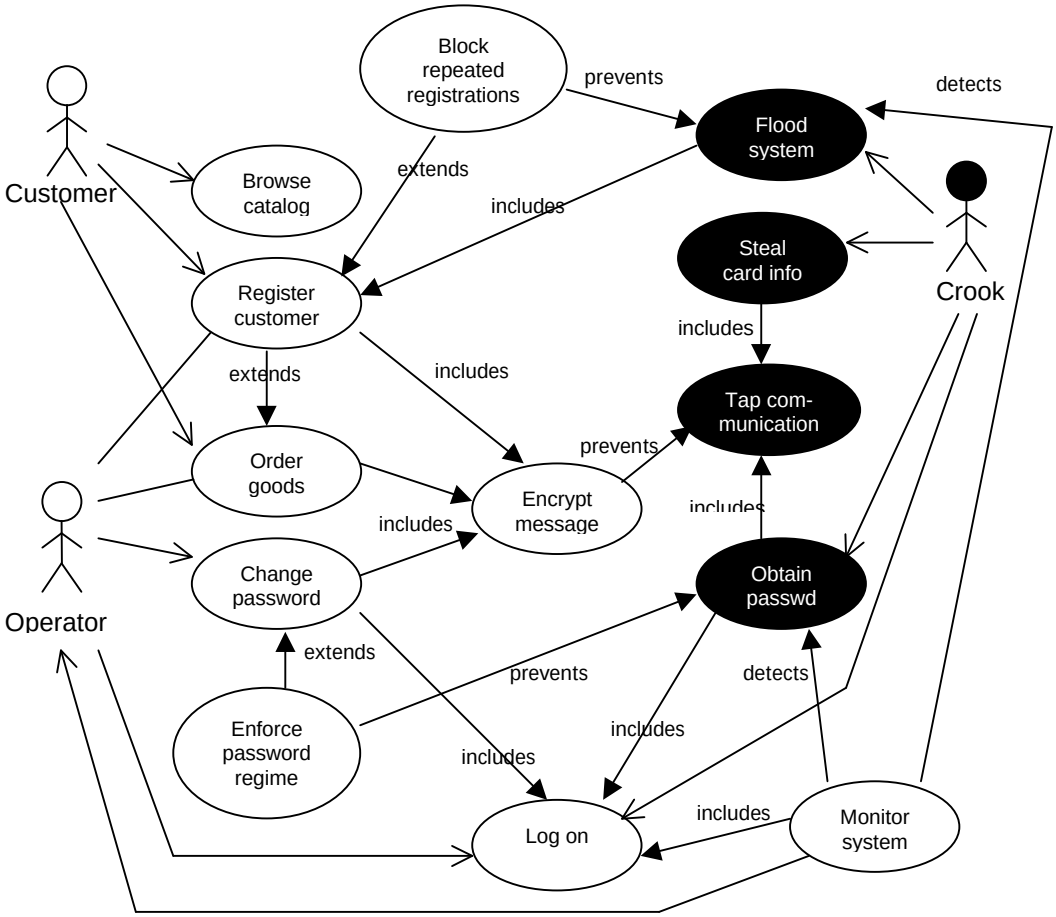

Figure 1: Misuse depicted as "inverted" use cases.

Apart from suggesting these concepts, [11] mainly looked at notation, as shown in Figure 1. This figures depicts part of the functionality for an e-commerce system. Misuse cases and mis-actors are inverted.

There may be various relations between ordinary use cases and misuse cases. First, there can be an "includes" or "extends" relation from a misuse-case to an ordinary use case, showing that some function for ordinary use is utilized for misuse. Two other relations were introduced in [11], "prevents" and "detects", which go from ordinary use cases to misuse cases, to indicate functions that prevent or detect misuse. Summing up the discussion of [11]: Because of these and other important relations between use and misuse, it is useful to be able to depict the two in the same diagram.

However, as stated in [3], the most important aspect of use cases is *not* the use case diagrams, but the textual representation of the use cases. That is the main topic of this paper, which is structured as follows: Section 2 reviews some often used templates for normal use cases. Section 3 then discusses adaptations of these templates to capture misuse cases. Section 4 outlines a method for using the new templates. Section 5

discusses the feasibility of the approach and its relation to other work. Section 6 concludes the paper.

2. Templates for use cases

Various templates have been suggested for the textual description of use cases. Kulak and Guiney [5] suggests a template consisting of the following parts:

- Use Case Name: A simple, intuitive name that uniquely identifies the use case.
- Iteration: Façade/filled/focused/finished —how refined the description is.
- Summary: One-two sentences describing the interaction occurring in the use case.
- Basic course of events: The steps that the actors and the system go through to accomplish the goal of this use case, the most common path taken.
- Alternative paths: Less common paths than the basic course.
- Exception paths: Paths taken when errors occur.
- Extension points: Steps in the use case from where extending use cases diverge.
- Triggers: The entry criteria for the use case, i.e., what initiates it.
- Assumptions: Conditions assumed to be true (but which cannot be guaranteed by the system itself) for the use case to be executed normally.
- Preconditions: Conditions that must be true before the use case can be performed.
- Postconditions: What will be true after the use case has been completed.
- Related business rules: Declarative business rules related to the use case.
- Author: Who wrote the use case.
- Date: When it was written.

The template suggested by Cockburn [3] has several interesting deviations from Kulak and Guiney's template, although the most evident fields are overlapping. Most notably, some new fields are introduced:

- Primary actor: Who initiates or performs the major actions in the use case.
- Scope: Here it can be distinguished whether the use case involves, e.g., the entire business or just the planned computerized information system.
- Level (of abstraction): This can be, e.g., summary, user goal, or sub-function.
- Stakeholders and interests: Listing the various stakeholders w/motivations.
- Technology and Data variations list: These are for various ways to do the same thing (i.e., if there are several *how's* for the same *what*, for instance successfully paying by cash, check or credit card). In Kulak and Guiney's template, such variations would end up as alternate paths.

In addition, there are some more minor variations which will not be dicussed here.

The template suggested by the Rational Unified Process [15] contains many of the same entries. Its basic form runs 1. Use Case Name, 1.1 Brief Description, 1.2 Actors, 1.3 Triggers; 2. Flow of events, 2.1 Basic Flow, 2.2 Alternative Flows, 2.2.1 Condition 1, (what to do), 2.2.2 Condition 2 …, etc.; 3. Special Requirements, 3.1

Platform …, …; 4. Preconditions, 5. Postconditions, 6. Extension Points. As can be seen, this is quite similar to Kulak and Guiney's template. The most notable deviation is the inclusion of a section for special requirements, such as platform requirements.

There are several other templates for use cases, but for the purpose of misuse cases it is not necessary to look at all these. The point is rather to see what fields are normally included in use case templates, and then to consider which of these would also be relevant for misuse case templates.

3. Adaptations for misuse cases

Some items that are common to all the three above-mentioned templates are obviously relevant for misuse cases, too:

- Name: Just like normal use cases, every misuse case should have a distinguishing name.
- Summary: One or two sentences should describe the interaction performed.
- Author: Whoever wrote the misuse case.
- Date: When it was written.
- Basic path: This will be the normal path of action taken, ending with success for the misuser and thus failure for the system and its owners.

For other items that also seem relevant, there are discrepancies between the three templates we looked at in the previous section. In the following discussion, which is a somewhat shortened version of the one found in [16], we underline (like above) the fields that are suggested as part of our new template. Tables 1 and 2 show how the fields are used to describe the "Obtain password" misuse case of Figure 1 in greater detail.

- Extensions/alternative paths/exceptional paths: Here Kulak and Guiney have three fields (alternative paths, exception paths and extension points.) Cockburn partly covers the alternate paths with his technology and data variations list, but in a more abstract manner. However, for misuse cases it may be of particular interest exactly to highlight specific technologies or extreme data values that can be exploited. Hence, we keep the slot for Alternate paths. In the context of misuse cases, exceptions/extensions are also meaningful, although in a less obvious way. This field is used to represent the various ways in which misuse is *prevented* or *detected*, cf. fig. 1. Because prevention and detection of misuse cases can other be tightly connected, we introduce a uniting field called *capture points* to cover both these. As Kulak/Guiney we also recommend a slot for *extension points*, showing optional actions which may be taken. Extension points will cover actions that the mis-user wants to perform, for instance to hack around hindrances. Capture points, on the other hand, work against the mis-user. Within the misuse case it might be appropriate to list various known or suggested options for dealing with the misuse – whereas a detailed description of how's and a choice of the exact defense mechanisms to implement would be a matter for design – following a more detailed analysis of risks and implementation costs which goes beyond pure (mis)use case modeling.

Preconditions:
pc1 The system has a special user 'operator' with extended authorities.
pc2 The system allows the operator to log on over the network.

Assumptions:
as1 The operator uses the network to log on to the system as operator (for all paths.)
as2 The operator uses his home phone line to log on to the system as operator (for ap2.)
as3 The operator uses his home phone line to log on to the system as operator (for ap3.)

Worst case threat (postcondition):
wc1 The crook has operator authorities on the e-shop system for an unlimited time, i.e., she is never caught.

Capture guarantee (postcondition):
cg1 The crook never gets operator authorities on the e-shop system.

Related business rules:
br1 The role of e-shop system operator shall give full privileges on the e-shop system, the e-shop system computer and the associated local network host computers.
br2 Only the role of e-shop system operator shall give the privileges mentioned in br1.

Potential misuser profile: Highly skilled, potentially host administrator with criminal intent.

Stakeholders and threats:
sh1 e-shop
- reduced turnover if misuser uses operator access to sabotage system
- lost confidence if security problems get publicized (which may also be the misuser's intent)

sh2 customer
- loss of privacy if misuser uses operator access to find out about customer's shopping habits
- potential economic loss if misuser uses operator access to find credit card numbers

Scope: Entire business and business environment.
Abstraction level: Mis-user goal.
Precision level: Focussed.

Table 2: Detailed misuse case description, part 2.

- Preconditions: Here, Kulak and Guiney have three somewhat related fields: *triggers* (entry criteria, what initiates the use case), *assumptions* (conditions which must be true but which cannot be guaranteed by the system itself) and *preconditions* (which can be ensured by the system itself.) The trigger field is needed to take care of situations where something else than the primary actor initiates the use case (such as timing, e.g., end of month). For misuse this may be interesting in connection with viruses, which might trigger on timing. Hence, we find it useful to keep the trigger field. Assumptions and preconditions are also kept, as they both seem relevant to misuse cases. Especially, there are useful for further security analysis, since breaking the assumptions or preconditions of a misuse case would prevent that particular kind of misuse from happening.

- Postconditions: Here, Kulak and Guiney has only one field, whereas Cockburn's postconditions are split into minimal guarantees (which will be true whatever path is taken) and success guarantees (which will be true when a no-error path is taken). Kulak and Guiney's postcondition field does not take height for error paths. For misuse cases we need something similar to postconditions/guarantees, but with slightly different names:
  - Worst case threat: Desribes the outcome if the misuse succeeds. If the misuse case has alternative paths, this condition will often be or contain a disjunction to describe slight variations in the outcome.
  - Prevention guarantee: Describes the guaranteed outcome whatever prevention path is followed. If no prevention path is followed, one might alternatively formulate a wanted prevention guarantee, expressing what one would want the system to achieve with respect the attempted misuse, but without stating how.
  - Detection guarantee: Describes the guaranteed outcome whatever detection path is followed. As above, one might also make this a wanted detection guarantee.
- Related business rules: Suggested as a separate field by Kulak and Guiney, to enter more declarative business rules related to the operation of the use case. We keep this field to link misuse cases to related business rules. This way, one can see exactly what business rules are broken by each misuse case — and possibly discover situations where the business rules themselves are too weakly formulated, thus opening for misuse. In this way, the business rule field provides important trace information when analyzing how to prevent misuse.
- Iteration: Obviously, there may be a need also with use cases to have both superficial and more detailed descriptions. However, this is a process issue which is not directly related to misuse cases. We therefore leave it open, recommending that the same description levels are used for misuse cases as for regular use cases.

Some of the fields suggested only by Cockburn [3] are also interesting to discuss:

- Primary actor: For a use case the primary actor will often be associated with a certain role in the organization. For a misuse-case the primary mis-actor may be a totally unknown with varying or unclear intent (economic gain, revenge, fun). Nevertheless, the field can be useful for stating whatever there is to state about the mis-actor. A more appropriate name is probably
  - Misuser profile. For instance, some kinds of misuse are most likely to be performed by intent whereas other may happen accidentally. Some require insiders or people with enormous technical skill, others not.

  Often, there may be several kinds of information to put in this field, so that it could be refined into several sub-fields. This, however, is not dealt with in this paper but left for further work.
- Scope: This field represents the scope of modeling, e.g., an entire business, an information system of users and computers, or just the computerized information system. Business level models can show how an outsider interacts with humans inside the organization instead of only showing how someone interacts with the system. This is important, because the techniques used to capture misuse attempts may be of a physical or organizational nature, in addition to designing security

into the computerized information system. Also, the computerized system is no more secure than its physical and organizational environment. The *scope* field is therefore considered important and kept as is.

- Level: This field indicates what level of abstraction a use case is on. This can be e.g., summary, user goal, or sub-function. Misuse cases can be specified at several levels of abstraction, so this field is kept.
- Stakeholders and interests: This field lists the various stakeholders and what their motivations are. For misuse cases this slot is actually even more important, but it might be renamed stakeholders and risks. On an abstract level, risks could simply be described textually (e.g., system unavailable for customers for several hours; potential employer gets sensitive data about patient, who thus loses job opportunity, …). With more ambition one might try to quantify costs (in terms of direct losses, badwill, …) and risks (likelihood that the misuse will occur). Here, conventional hazard analysis techniques would come into play to determine which misuse cases are the most important to prevent (weighting the risk*loss against the cost of the prevention strategy), but hazard analysis is not a topic of this paper. The interplay between this and misuse cases is thus left to further work.
- Technology and data variations: This list is also interesting for a misuse case template. Although there is some overlap with the alternate paths discussed earlier, the two are not exactly the same. With the technology and data variations list, it is possible to mention some variations without giving a path for each of them. E.g., a misuser may perform a certain misuse case accessing the system either through a PC web browser or a WAP phone, but apart from detailed actions related to the particular equipment, the misuse in these two cases will be the same. In this case it may feel unnecessary to list two separate paths for this, instead the various equipment may simply be listed.

4. Method guidelines

A detailed discussion of how to go about misuse case modeling is beyond the scope of this paper, which is mainly about templates. However, some points will be important to discuss in this context:

- The use of textual description vs the use of diagrams.
- How to gradually extend the textual descriptions with more detail, and yet avoid to make premature design decisions.

With respect to the first question, there may be two opposite opinions:

1. Start with diagrams, then supplement these with text providing more detail.
2. Start with text only, introduce diagrams when the number of use cases grow big enough to mandate the need for overview pictures.

Both opinions make sense. It is often thought a useful way of modeling to begin with an overview and then provide more details. But the contenders of the second (like [3]) might say that when it comes to use cases, the diagrams are really too vague to say anything. Thus, these are only useful as an overview facility — which is pointless when you have defined only one or two use cases. Possibly, diagrams could be generated automatically from the textual descriptions. Or, if going the other way, at least part of the templates could be filled in automatically based on diagram content.

Although we have sympathy with the arguments of [3] that use cases are mainly a textual kind of modeling and that use case *diagrams* have been oversold by the tool industry, we do not see the need to take a definite stand on this issue. Rather, a project team should probably use the same sequencing for misuse cases as for use cases. If they are accustomed to going diagrams then text with use cases, the same should be done for misuse cases. And if they are accustomed to text only or text then diagrams for their use cases, the same process should be followed for misuse cases.

The second question is more vital. We have suggested a template with quite a number of fields, which will often seem over-elaborate early on, when one is just trying to get an overview of various use- and misuse cases. However, the template is just a suggestion for fields that a misuse case description should contain in its finished state. Along the way, most of the fields would not be mandatory. Just like a use case, a misuse case could be presented with only a name and brief description, along the lines of the "casual" version suggested in [3] or the "facade" iteration suggested in [5], allowing for what is traditionally called specification freedom [17]. Something more about sequencing:

1. First concentrate on the normal actors and the main use cases requested by these, i.e., the services that the users want, regardless of any security considerations — as suggested in your chosen method (whether this be the one of RUP, Cockburn, Kulak and Guiney or something else). The motivation for this is quite simple: If there is nothing to use, there is nothing to misuse — hence it would be difficult to start with misuse cases.
2. Then introduce the major mis-actors and misuse cases, i.e., threats that are likely. If possible, there could be more mis-actors than given in our examples, and they could be more precisely named. E.g., if the procuring company has a special concern that their competitors might sabotage the system, one mis-actor could be called competitor. This would give a clearer picture of that mis-actor's motivation.
3. Investigate the potential relations between misuse cases and use cases, in terms of potential "includes"-relations. This step is quite important since many threats to a system can largely be achieved by using that system's normal functionality, as for instance the "Flood system" misuse case of our example.

However, we do not generally recommend early investigation of prevents or detects relations, especially not step descriptions of these — which would easily entail premature design decisions. Although it may be hard to distinguish clearly between requirements and design in many cases, one rough guideline might be that

- *how* some misuse is prevented is design, whereas
- the fact that the customer does not want it to happen, is a requirement.

This is actually the nice thing about misuse cases — it makes it possible, so to speak, to capture requirements in "negative space". Without the option of registering misuse cases as such, the modeler would have to model countermeasures directly (at least in a use case context), forgetting that these are normally design decisions. With misuse cases one can simply state that "this is something that should not be possible" and then leave it to design to decide exactly how the threat is avoided. A valuable input to

this design decision would be various known or suggested defenses listed in the "capture points" field of our template, which could then be analyzed more thoroughly for coverage and cost [18]. When the relations between misuse cases and normal use cases have been documented, it is also possible to say something about the security cost of various functions that we *want* the system to perform, which is important for prioritization. For instance, the end users may have requested a use case UC1 which is useful but not strictly necessary for their work. Assuming that UC1 as such is quite easily implemented, it would be tempting to include it in the system. But imagine, then, that UC1 enables (through some misuse, or through common assumptions) a really dangerous misuse case MUC1 – for which a really costly protection mechanism might have to be built into the system. Then the real cost of UC1 is not only its own implementation, but also the protection against MUC1. Here, misuse case analysis might illustrate that it could be wise to drop UC1 from the system.

Most of our discussion here suggests that misuse case analysis will be used early on in the development process. But it might also be an idea to redo misuse case analysis on a more detailed level *after* the system's security defenses have been chosen (or preferably, tentatively chosen) – this may provide an opportunity to paper test the choice of defenses and try to find weaknesses. As stated in [12], when some defenses are introduced, potential attackers will look for other openings – and may find some where it was quite unsuspected. Applying misuse case analysis on several levels of abstraction and at several stages during the development process may increase the chance of eliminating such unpleasant surprises.

5. Discussion

The proposed template for detailing misuse cases supports representation of information relevant for security considerations during requirements determination. The proposed approach could also be helpful in the early elicitation of security requirements in several other ways:

- Early focus on security. Security issues are too often considered a design-level — and sometimes even primarily a programming-level — issue. This is unfortunate, because the requirements stage is where the appropriate security issues can be identified, analyzed and balanced against one another. Indeed, many security issues will be most easily captured by physical or organizational means, rather than in the planned information system. When security considerations are postponed to design and programming, such security issues will easily be forgotten. Even when they are taken into account, superior physical or organizational solutions may be ignored in the design phase.
- User/customer assurance. Although the end-users and management of the procuring organization may not be able to discuss the technical details of security threats and countermeasures, they will have concerns about various threats. Seeing these captured by misuse cases will reassure them that the problem is actually being dealt with.
- Analyst creativity. Computer criminals are surely going to be creative in their attempts to fool the system. Thus analysts need to be equally creative to identify the relevant threats beforehand. When explicitly writing misuse cases, analysts may get ideas about possible misuses not thought about before.

- Traceability. If describing the countermeasures as first-class requirements, one easily misses out on the motivation for these countermeasures. The explicit registering of misuse cases can show why a certain defense was introduced in the system, which will help later maintenance. Assume, for instance, that a certain kind of security attack was initially believed to be fairly unlikely. If the best defenses were expensive, one may have decided to go for a cheaper but weaker defense. However, after the release of the system, a thorough recipe for how to perform such an attack is published on some cracker web site – which suddenly increases the risk considerably. If the misuse case was explicitly modeled, one need only adjust the believed risk of that misuse case and then follow links to any part of the design relevant for that misuse case to review these. Plus, the more expensive defense options might already be listed as capture points in the misuse case, so that there are clear hints on how to proceed.
- Requirements organization. A major problem in managing security and other extra-functional requirements is how to organize the overall requirements document and, in particular, how to relate functional and extra-functional requirements to one another. Misuse cases solve this problem for a large class of extra-functional requirements. As mentioned previously, one useful situation to detect is where a use case has the same assumptions and preconditions as a misuse case. This is a clear hint to investigate whether that use case is strictly necessary, as its removal might make it much cheaper to defend against the misuse case. Such analysis might resemble the use of root requirements (that other more detailed requirements are derived from) in [18].
- Transition to object-oriented design. An additional advantage, of course, is that the application of use cases also for security-intensive systems, makes it possible to build on all the ongoing work providing transitions between this and object-oriented design
- Uniform handling of functional and extra-functional requirements. As mentioned in the introduction, misuse cases are applicable and useful for dealing with a larger class of extra.-functional requirements which includes security requirements. Extra-functional requirements dealing with *safety*, *availability* and *robustness* all state what should *not happen* and therefore fall into this class. As a consequence, misuse cases offer a uniform way of handling them. This is significant, because a major obstacle to dealing with extra-functional requirements is their inherent diversity, which calls for a broad array of techniques to be used in requirements work, but which makes the resulting requirements document difficult to organize and comprehend.

There are several methods that somehow utilize use cases or scenarios in requirements capture, for instance [20,21,22,23]. On the other hand, there are methods suggested for security requirements engineering that are not use case based, e.g., [24]. In [25] there is a discussion of using scenarios for developing secure e-commerce systems, but not suggesting any concept resembling the misuse case. Hence, this to be a novel concept, and our investigations indicate that it can be a quite useful extension to the repertoire of the requirements engineer.

On the other hand, misuse case analysis alone makes no requirements engineering technique. Methods such as the ones referred to above have many years of work behind them and are a lot more detailed than our misuse case analysis with respect to method guidelines and tool support. We only look at misuse analysis where other

methods address scenarios/use cases and goal structures in general. Hence, the authors plan to investigate how misuse case analysis can be integrated with existing RE methods and techniques.

6. Conclusions and further work

Standard use case diagrams are often good for eliciting functional requirements, but not so good for security requirements, which relate to activities *not* wanted in the system. In a previous paper we have suggested some small extension to use case concepts and diagrams, and in this paper a textual template for misuse cases has been discussed in more detail. The approach has been tested on several small examples and is used in ongoing research investigating the use of *security patterns* in requirements work. However, misuse cases must now be evaluated in an industrial setting. We think the proposed approach is well-suited for such evaluations:

- The approach is close to standard UML use case diagrams, which are already heavily used in industry.
- Security requirements are increasingly important with the advent of e- and m-commerce, and the problem that use case diagrams deal poorly with these, *will* be an industrial problem.

The suggested extensions to use case diagrams are small, and simple to implement in a tool prototype, and the textual misuse case template builds on well-known templates for normal use cases. Thus, we believe that the notation and template for misuse cases suggested in this paper can fairly easily be incorporated in a software development organization where use case analysis is already applied. As mentioned above, the authors plan to look further into possible integration with other use case based and goal based RE approaches. Also, work is in progress on establishing a library of such *security misuse case patterns,* and using them for eliciting security requirements in practical settings. Other interesting directions to pursue are integrations with techniques for risk analysis, costing, and formal methods, which have been popular in security and safety engineering. Safety might in itself be an interesting direction to widen the application of misuse cases – again to describe things that should not happen in a system. As stated by [26], there are several interesting connections between safety and security, and [27] observes that a better integration between informal and formal techniques is needed to make progress in RE in this field.

[1] I. Jacobson et al., Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1992.

[2] L. L. Constantine and L. A. D. Lockwood, Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design, ACM Press, 1999.

[3] A. Cockburn, Writing effective use cases, Addison-Wesley, 2001.

[4] J. Rumbaugh, “Getting Started: Using use cases to capture requirements”, Journal of Object-Oriented Programming, September 1994, pp. 8-23.

[5] D. Kulak and E. Guiney, Use Cases: Requirements in Context, ACM Press, 2000.

[6] M. Arnold et al., "Survey on the Scenario Use in Twelve Selected Industrial Projects", technical report, RWTH Aachen, Informatik Berichte, Nr. 98-17, 1998.

[7] K. Weidenhaupt, K. Pohl, M. Jarke, and P. Haumer, "Scenario Usage in System Development: A Report on Current Practice", IEEE Software, 15(2): 34-45, March/April 1998.

[8] A. I. Antón, J. H. Dempster, and D. F. Siege, “Deriving Goals from a Use Case Based Requirements Specification for an Electronic Commerce System”, Proc. REFSQ’2000, 5-6 Jun 2000.

[9] J. Arlow, “Use Cases, UML Visual Modelling and the Trivialisation of Business Requirements”, Requirements Engineering Journal, 3(2):150-152, 1998.

[10] S. Lilly, “Use Case Pitfalls: Top 10 Problems from Real Projects Using Use Cases”, Proc. TOOLS-USA’99, pp.174-183, 1-5 Aug 1999.

[11] G. Sindre and A. L. Opdahl, "Eliciting Secutiry Requirements by *Mis*use Cases", Proc. TOOLS Pacific 2000, pp 120-131, 20-23 Nov 2000.

[12] C. P. Pfleeger, Security in Computing, Prentice-Hall, 1997.

[13] G. Kotonya and I. Sommerville, Requirements engineering: Processes and Techniques, Wiley, 1997.

[14] P. Loucopoulos and V. Karakostas, Systems Requirements Engineering, McGraw-Hill, 1995.

[15] P. Kruchten: The Rational Unified Process – an Introduction, Addison-Wesley, 2000.

[16] G. Sindre and A. L. Opdahl, "Templates for Misuse Cases", Proc. REFSQ’2001, Interlaken, Switzerland, 4-5 June 2001.

[17] P.E. London and M.S. Feather, “Implementing specification freedoms”, Readings in Artificial Intelligence and Software Engineering, pp. 285-305, Morgan Kaufmann, 1986.

[18] C. Irvine, T. Levin, “Toward a Taxonomy and Costing Method for Security Services”, Proc. ACSAC’99, Phoenix AZ, 6-10 Dec 1999.

[19] W. N. Robinson, S. Pawlowski, “Surfacing Root Requirements Interactions from Inquiry Cycle Requirements Documents”, Proc. ICRE’98, Colorado Springs, 6-10 Apr, 1998.

[20] A. I. Antón, “Goal-Based Requirements Analysis”, Proc. ICRE’96, pp. 136-144, 1996.

[21] N. Maiden, S. Minocha, K. Manning, and M. Ryan, “CREWS-SAVRE: Systematic Scenario Generation and Use”, Proc. ICRE’98, pp.148-155, 1998.

[22] C. Potts, “A ScenIC: “A Strategy for Inquiry-Driven Requirements Determination”, Proc. RE’99.

[23] C. Rolland, C. Souveyet, and C. Ben Achour, “Guiding Goal Models Using Scenarios”, IEEE Transactions on Software Engineering, 24(12): 1055-1071, Dec 1998.

[24] J. Kirby Jr, M. Archer, C. Heitmeyer, “SCR: A Practical Approach to Building a High Assurance COMSEC System”, Proc. ACSAC’99, Phoenix AZ, 6-10 Dec 1999.

[25] A. I. Antón, J. B. Earp, ”Strategies for Developing Policies and Requirements for Secure Electronic Commerce Systems”, Proc. 1st ACM Workshop on Security and Privacy in E-Commerce, Athens, 1-4 Nov 2000.

[26] D. M. Berry, “The safety requirements engineering dilemma”, Proc. 9th Int. Workshop on Software Specification and Design, 1998.

[27] R. R. Lutz, “Software Engineering for Safety: A Roadmap”, in A. Finkelstein (ed.): The Future of Software Engineering, ACM Press, 2000.