Meta-models for Misuse Cases

Mark Hartong
Federal Railroad Administration
1200 New Jersey Ave SE
Washington, DC 20590
+1 202 493 1332
mark.hartong@dot.gov

Rajni Goel
Howard University
2600 6th Street South
Washington, DC 20590
+1 202 860 1649
rgoel@howard.edu

Duminda Wijesekera
George Mason University
100 University Drive
Fairfax, VA 22030 DC
+1 703 993 1578
dwijesek@gmu.edu

ABSTRACT
Misuse Cases are recent UML constructs that can be used to specify the mal-acts against which a requirements engineer seeks guarantees from the designer. However, Misuse Cases have not been formally adopted in UML, and therefore lack a formal meta-model. This paper proposes a meta-model that covers graphical, textual and OCL models for Misuse Cases that augments the existing UML 2.0 Use Case meta-models.

Categories and Subject Descriptors
A.2 [General Literature: Reference]:

General Terms
Documentation, Design, Standardization

Keywords
Use Cases, Misuse Cases, UML, OCL

1. INTRODUCTION
The Unified Modeling Language (UML) [1] is a commonly used throughout the software development life cycle. Use Cases – used in the requirements specification stage of the life cycle - are one construct used in UML to specify the interactions that the system is supposed to carry out with the external entities (so called actors). Consequently, a system that is developed using Use Cases provides all those interactions. Naturally, Use Cases assume that the actors interacting with the system are not acting with malicious intent.

Such an assumption does not adequately reflect the real world, where systems are often under attack by malicious entities, called mal-actors, who are intent on disrupting the intended use of the system. In order to specify such actions, Misuse Cases have been proposed [2]. Due to short time period Misuse Cases have been around, they have not been formally adopted in UML and therefore lack formal definition.

2. PRIOR RESEARCH
Misuse Cases provide a UML mechanism for capturing the relationships between actors and mal-actors and their impact on the behavior of the system. As first defined, misuse cases add two additional constructs to UML: <<prevents>> and <<detects>>. In the <<prevents>> relationship, the functionality of a misuse case or mal-actor prohibits the execution of the functionality of the related use case. In the <<detects>> relationship, the construct reflects the functionality of a use case discovering a misuse case or mal-actor capable of preventing the execution of the use case.

These two misuse relationships were further extended by Alexander [3] to include <<threatens>>, <<mitigates>>, <<aggravates>>, and <<conflicts with>>, resulting in an increase in the descriptive power of Misuse Cases. In the <<threatens>> relationship, the functionality of the Use Case is not eliminated by the actions of the mal-actor, as is the case with <<prevents>>; rather, there is placed in jeopardy from executing properly. In the <<mitigates>> relationship, the mitigating use case counteracts the action of a mal-actor or another misuse case. The functional inverse of <<mitigates>> is <<aggravates>>. In the case of <<aggravates>>, the aggravating misuse case worsens the adverse impact of an existing adversarial relationship. Finally the <<conflicts with>> relationship states the mutual exclusivity of the related use cases.

McDermott and Fox have also proposed a modeling construct called Abuse Cases that are similar to misuse cases [4] to state real world behaviors of systems subjected to adverse influences. However, abuse case models differ from misuse case models in two ways. First, abuse cases are not shown on a use case diagram and use cases are not shown on an abuse case diagram. This is in contrast to misuse cases that are blended with the use case models, forming an integrated statement. Second, abuse cases, by definition, are limited to eliciting security requirements, unlike misuse cases that are intended to elicit any negative relationships. However, sans intent, misuse cases and abuse case are synonymous.

Significant effort has already been expended in defining formal meta-model translations from many UML constructs. While formalizations of Use Cases based on automata have been proposed [5], their abstract nature is generally unsuitable for use by those without a strong background in automata theory. Other formalizations, for example, using Z [6] and Larch [7], are available for describing Use Cases. Although these formal languages have great expressive power, they are formal abstract notations, and, like formal automata, require a high degree of mathematical maturity to use effectively. Conversely, the Object Constraint Language (OCL) [8] is a fragment of first order logic tailored for UML. When combined with Operational Schema, OCL provides a formal and mathematical syntax for further stating the constraints related to Use Cases.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CSIIRW '09, April 13-15, Oak Ridge, Tennessee, USA Copyright 2009 ACM 1-58113-000-0/00/0004...$5.00.

The views and opinions expressed herein are those of the authors and do not necessarily state or reflect those of the United States Government, the Department of Transportation, or the Federal Railroad Administration, and shall not be used for advertising or product endorsement purposes.
3. GRAPHICALLY REPRESENTING UML MISUSE CASES

The purpose of defining Misuse Cases in terms of extensions of already defined UML 2.0 conventions is to (1) to enable the created models to be readily understandable to UML users, and (2) to be able to manipulate those models using standard UML tools.

Graphical representations for Use Cases have been formally defined in [1]. The primary notation for an actor is a stick figure with a short descriptive name. The common notation, used both by Alexander [3] as well as Sindre and Opdahl [2], for a mal-actor is a stick figure with a short descriptive name. This is the primary notation for the mal-actor. Unlike the actor, however, the mal-actor’s head is solid black, as opposed to clear. The use of colour’s to assist in the interpretation of UML objects is not with precedent. Couad et al [9] suggested the use of “pink”, “yellow”, “blue”, and “green” to represent time dependent, role dependent, simple descriptions, and other elements. Although not specified by either [2] or [3], but in keeping with standard UML 2.0 [1], the mal-actor will also be allowed to be shown using a stereotyped class, or a user defined icon. (Figure 1)

The primary representation in standard UML 2.0 for the Use Case is an ellipse, with a short name that contains an active verb and noun phrase either in, or below, the ellipse, followed optionally by a list of properties. Also, as shown in [2] and [3], and similar to the mal-actor, the Misuse Case will be defined as a solid ellipse with a short name containing an active verb and noun phrase either in, or below, the ellipse. This is extended to include an optional list of properties as specified in UML 2.0.

The relationship of mal-actors and misuse cases follows a similar pattern to that of actors and use cases, just as actors are external entities to the system and interact only with use cases, mal-actors are external entities to the system and only interact with misuse cases. The normal associations between an actor and a use case, a use case and a use case, a mal-actor and misuse case, and a misuse case and misuse case follow standard UML 2.0 notation. Solid lines represent these associations. Unlike [2] and [3], which also use solid lines labeled with the nature of the misuse relation (i.e., <<prevents>>, <<detects>>, <<threatens>>, <<mitigates>>, <<aggravates>>, and <<conflicts with>>) to define associations between use cases and misuse cases, our formal definition follows that of the <<includes>> and <<extends>> relations. Specifically, a dashed arrow represents the relationship, with an open arrowhead from the casual use case/misuse case to the use case/misuse case impacted by the causing use/misuse case. These are all binary associations as required by standard UML.

Figure 1: Recommended Symbols for Mal-actors and Misuse Cases

The six associations defined for use/misuse cases relate specific actor/use case and mal-actor/misuse causal-caused action case pairs. (Figure 2) For example, based on the definition in [2] a use case <<prevents>> the activation of a misuse case, while a use case <<detects>> the activation of a misuse case. In the <<threatens>> relation, the misuse case reduces the likelihood of the use case functions from completing. Consequently completion of the misuse case is considered an undesirable event, though threatening a misuse case is not considered an adverse event. With the <<mitigates>> relation, the use case reduces the likelihood of the Misuse Case function affecting the Use Case it threatens. Use Cases do not <<mitigate>> Uses. Misuse Cases do not <<mitigate>> Misuses, and Misuse Cases do not <<mitigate>> Use Cases. The former because Use cases are presumed not to have a negative impact on other Use cases, the later because Misuse Cases prevent a Use Case function rather permit a Use Case function.

Not previously specified in the definition of Misuse Cases, but following UML 2.0, a note can be attached to all relationships. The note is attached to the association arrow, and may contain amplifying information regarding any constraints or other relevant information associated with the relationship, and could include affected attributes, information that is pertinent to their instances, and operations or functionality associated with the affected case. The formalized definition of Misuse Cases presented also adopts the standard UML 2.0 Use Case multiplicities of the actors, mal-actors, use cases, and misuse cases.

4. TEXTUALLY SPECIFYING MISUSE CASES

While compelling, graphical representations of use cases, misuse cases, and their associations have a significant drawback that is caused by the graphical representation not capturing the complex nuances and details of textual descriptions. Consequently textual descriptions of use cases and misuse cases are necessary, and these must specify details such as sequential behavior within a binary relationship, parallel behavior between binary relationships etc.

Different templates have been proposed for use case descriptions, for example [10, 11, 12, 13]. Each of these recommends various styles, content, and formats, along with different approaches for developing the material to be specified. The templates for Misuse Cases have been well developed [14], and are adaptable to either the “causal” development approach of [10] or the “façade” development approach of [11].
### Figure 3: Standard Misuse Case Template

The fields in Figure 3 are generally self-explanatory. “Name”, “Summary”, “Author”, and “Date” represent a unique name for the misuse case, a summary of what system functionality the misuse case disallows, the definer of the misuse case, and the date the misuse case was prepared. The “Basic Path” describes the usual course of events that results in the misuse “succeeding” and the system functionality being disallowed.

“Alternate Paths”, “Capture Points”, and “Extension Points” provide alternate courses of events that result in the misuse “succeeding” and the system functionality being disallowed, along with constraints/limitations or enhancements that affect the execution of the alternate paths. “Capture Points” are negative constraints on the misuse case, they specify how the alternate path may be prevented or detected. “Extension Points” are conditional enhancements that positively affect the successfully execution of the misuse case.

“Triggers”, “Preconditions”, and “Assumptions” establish the necessary and sufficient conditions for the misuse to occur. “Triggers” identify all conditions that are responsible for initiating the misuse case. “Triggers” differ from “Preconditions” and “Assumptions”, in that preconditions and assumptions reflect the state of the system, rather than actions upon the system. “Assumptions” are states of the system that must be true, but cannot be identified as being true (usually environmental factors), while “Preconditions” are states of the system that must be true, and can be identified as being true (usually system configuration factors).

The “Worst-case Threat” describes the result if the basic or any of the alternate paths were to succeed without any detection or mitigation. “Prevention Guarantee” describes the result that would be obtained by following the “Basic” or “Alternate” path if the threat were detected and minimum threat mitigation occurs. The “Detection Guarantee” describes the result that would be obtained if the misuse were detected, but not mitigated. “Related Business Rules” are use case business rules that are broken by the misuse case.

The final fields are “Iteration”, “Misuse Profile”, “Scope”, “Level of Abstraction”, “Stakeholder & Risks”, and “Technology & Data” variations. “Iteration” is the level of maturity of the specification of the misuse case (for example in the notation of [Kull90] this would be façade, filled, focused, finished). “Misuse Profile” provides descriptive information about the known characteristics of the mal-actor(s). “Scope” is the level of abstraction that the model covers, and is necessary to place the results of the model in context. The “Level of Abstraction” provides the level of the fidelity of the model, which should not be confused with the level of accuracy of the results. Because a model may accurately reflect the static complexity of the domain, but fail to totally capture the dynamic relationships. “Stakeholder & Risk” identifies interested parties affected either directly, or indirectly) by the misuse and its impact. Finally “Technology and Data” variations are for the identification of similar technology that can result in the same misuse result occurring.

Critical trait attributes are captured from the textual misuse cases from an analysis of nouns and verbs by a technique called noun-verb extraction [15]. By using noun-verb extraction analysis, specific characteristics that could represent evidence (i.e. behaviors that may be directly observed or conclusively inferred from observed behaviors) or invariant elements are identified. This process can be done entirely by hand by an engineer, or through the aid of tools [16].

### 5. SPECIFYING MAL-OPERATIONAL SCHEMA USING OCL

The graphical notation for misuse cases, which illustrates the names of the actors and mal-actors and their relationships to the use and misuse cases, as well as the textual descriptions, which more comprehensively describes the details of those relationships are informal specifications of the system behavior. Consequently, they are inadequate to precisely define how a system is expected to operate. OCL and Operational Scheme [8, 17], on the other hand, precisely define a systems interface and the functionality of the system.

#### Operation:
- Description:
- Notes:
- Use Cases:
- Scope:
- Declares:
- Sends
- Preconditions:
- Post-conditions:

### Figure 4: Standard Operational Scheme Template

“Classifiers”, “Attribute’s”, “Associations”, “Operations”, and “Enumerations” represent the elements of use case class models expressible in OCL. “Classifiers” are specific types of items. “Attribute’s” describe properties of the various classifiers, while “Associations” define the relationships between two or more classifiers. “Operations” are actions on classifiers that do not have a side effect, and “Enumerations” are extensions of the basic OCL types (specifically Boolean, Integer, Real, and String). An important distinguishing factor with OCL and Operational Schema Models from traditional graphical or textual use case models is the formalization in terms of constraints. Each of the elements is formally declared in terms of non-executable typed expressions that follow OCL syntactical rules. [8]. We will use the same syntactical rules for misuse cases.

The application of OCL and Operational schema as descriptors of traditional Use Cases is explored by Sendell and Stronheimer...
7. REFERENCES


[16] [Benot, Overmyer, Rambow “Conceptual Modelling Through Linguistic Analysis Using LIDA”, , Proceedings of 23rd International Conference on Software Engineering (ICSE 2001), Toronto, Canada


[17], primarily for the use case relationships of <<includes>>, <<extends>> and <<specialization>>. They define a standard template for use with Operational Schema, utilizing the syntactical elements of the formal OCL model while optimizing semantic interpretability. While the template is not part of OCL, the contents of the individual fields of the template are written using OCL. This same basic template is applicable to misuse cases. The mal-operation, as opposed to operation, field defines the system name, along with misuse operation name and the parameter list.

Translation of a misuse case in terms of OCL mal-operational schema is basically a four-step process. First is analysis of the particular misuse case model to identify the appropriate classifiers. Once the classifiers have been identified, the associations can be determined from the various relationships in the misuse case. This is followed by definition of the constraints, on the classifiers and associations, specified in terms of pre and post conditions. Finally invariant conditions are identified. This analysis allows completion of an Operational Scheme template for the misuse case. This same analysis process can be used to define mal-operation schema to support the Misuse Relationship operations <<prevents>>, <<detects>>, <<threatens>>, <<mitigates>>, <<aggravates>>, and <<conflicts with>>.

6. CONCLUSIONS
While each description captures different facets of the misuse, the combination of graphical diagrams, narrative descriptions and OCL based operational scenarios provides a comprehensive definition of the misuse case. This facilitates specifying system functional, behavioral, and communications relationships.

Meta-modal formalization of Misuse Cases, and their integration into existing Use Case definitions, is essential to make effective of Misuse Cases in specifying the mal acts against which it is expected to be robust. The schema presented here fully integrates with traditional Use Case meta-models. The proposed schema relies upon extensions of known informal and formal techniques to specify requirements and their inter-relationships - resulting in reducing ambiguity, inconsistency, and enhances completeness of the requirements.

Although the schema reduces ambiguity, incompleteness, and inconsistency, the schema could be further improved by reduction of the semi-formality. The semantics of the proposed schema still rely upon either completely informal (graphical and textual) notational representations or semi-formal (OCL) representations. While the semantic imprecision of graphical and textual representations is self-evident, OCL lacks a complete formal semantic definition [18]. This shortcoming is the basis for our ongoing studies for a more precise formalization of misuse cases using finite automata..