Safety and Security Analysis of Object-Oriented Models

K. Lano, D. Clark, K. Androutsopoulos[1]

Department of Computer Science, King's College London, Strand, London WC2R 2LS, UK. Phone: +44 (0)207 2832. Fax: +44 (0)207 2851. Email: david@dcs.kcl.ac.uk

Abstract. In this paper we review existing approaches for the safety and security analysis of object-oriented software designs, and identify ways in which these approaches can be improved and made more rigorous.

Introduction

Object-oriented software is increasingly being used for critical systems: the widespread use of Java for smartcard based security-critical applications in particular is one example where intensive verification and proof techniques have been used [5, 7, 4]. C++ has also been used in safety-critical and safety-related systems, and it seems likely that the use of such languages and methods will increase, as the impact of the standardisation of UML [11] and the convergence of commercial software development on Java and Java-like environments throughout the software industry spreads into the safety-critical systems domain. There are good engineering reasons for this trend, in that the capabilities of developers seem to be generally enhanced through the use of such methods and languages: it becomes easier to analyse and review the software and to detect and correct errors, because of the increased opportunities for modularising and structuring the software into relatively independent subcomponents with restricted interfaces. Java in particular could be regarded as a safe(r) subset of C++, eliminating many of the features, such as pointers and programmer defined memory management, which are a frequent cause of subtle errors in C and C++ programs. It would be direct to adapt the MISRA C rules [3] to Java, indeed many of these rules are already common practice for Java programmers. Assertion-based static analysis tools for Java are now available [12].

In this paper we examine how safety analysis techniques such as HAZOPS and fault tree analysis (FTA) can be adapted to object-oriented system and software modelling notations, particularly UML.

1 Hazard Analysis of Object-oriented Models

Standards such as [9] recommend that hazard identification, hazard analysis and risk analysis should be carried out for critical systems. Recommended techniques for these processes include HAZOPS [10] and Fault Tree Analysis (FTA).

HAZOPS consists of a process of systematic exploration of analysis and design diagrams and documentation of a system for possible deviations from required behaviour or design intent, and identification of hazards caused by such deviations. Based on a design diagram, the elements of the system are identified, and for each element attributes are identified, for example, the position of a valve. A set of guidewords such as *NO*, *REVERSE*, *AS WELL AS* are then applied to each attribute to identify possible deviations (eg, the valve being open when it should be closed). HAZOPS can be applied at an early stage of system construction, where the main features of a system and intended behaviour are known, but detailed algorithms or decomposition of modules have not been produced. FTA derives a detailed causality structure for each credible hazard produced from HAZOPS, and assigns maximum acceptable probability bounds to events based on the maximum acceptable probabilities of their consequent hazards.

Def-Stan 00-58 [10] gives some guidelines for the hazard analysis of state transition diagrams and entity-relationship-attribute models using HAZOPS. These guidelines and the examples given are unclear and ambiguous in some cases, which we discuss below. We provide an alternative and systematic set of guidelines for HAZOPS of UML notations, and give examples of their use.

1.1 State Transition Diagrams

For state transition diagrams Def-Stan 00-58 Part 2 considers the transitions as the basic elements of analysis and the attributes *event* and *action* of transitions. For events there are the following guidewords and interpretations (Table 1).

Guideword	Interpretation
No	Event does not happen
As well as	Another event takes place as well
Other than	An unexpected event occurs instead of the expected event

Table 1. 00-58 Guideword Interpretations for Event

However "Event does not happen" is ambiguous – its meaning depends on what the state machine model represents: a specification of a control system, or a description of the equipment under control (EUC). In the former case the guidewords can be alternatively interpreted as follows (Table 2).

For example, having commanded a railway switch (points) to change position, the control system expects an event confirming the completion of the close. Suitable action must be taken if this event does not occur within a given time period. An initial HAZOPS would consider deviations at the complete system level, not examining details of the control system behaviour but instead issues of how deviations from design intent in the controlled system can lead to hazards. For example failure of the switch to close can lead to derailment if undetected.

Guideword	Interpretations
No	Event not received by control system: either it occurs but is not transmitted to the controller because of sensor or other failure, or it does not occur even though expected
As well as	Another event is detected by the control system as well as the intended event
Other than	An unexpected event is detected instead of the expected event

Table 2. Revised Guideword Interpretations for Event

At an initial design level HAZOPS can consider the intended state transition behaviour of the control system and identify more specific ways that deviations from design intent can occur. Eg: the switch may actually close but the control system not detect this due to faulty sensors, connections or control logic, etc.

A similar distinction is necessary on the output side of a state machine as well. The attribute in this case is the transition *action*, which has the following guidewords (Table 3).

Guideword	Interpretation
No	No action takes place
As well as	Additional (unwanted) actions take place
Part of	An incomplete action is performed
Other than	An incorrect action takes place

Table 3. 00-58 Guideword Interpretations for Action

Again we can regard the above as appropriate when considering the overall behaviour of a system, and the following as appropriate when considering deviations of a software control system (Table 4).

Guideword	Interpretations
No	No action is produced by the controller, or this action is not transmitted to/carried out by actuators
As well as	Additional (unwanted) actions are generated/performed
Part of	An incomplete action is generated/performed
Other than	An incorrect action is generated/performed

Table 4. Revised Guideword Interpretations for Action

For states, deviations could be as given in Table 5.

Consideration of timing also needs to be relative to either the controlled system or the controller behaviour. A significant time delay between a real-world event and its detection by a controller would be a deviation considered by the latter analysis for example, whereas a railway switch closing more slowly

Guideword	Interpretation
No	Object not in expected state
Other than	Object in an unexpected state

Table 5. Guideword Interpretations for State

than intended, because of ice, for example, would be a deviation considered by the former.

1.2 Class Diagrams

Entity-relationship diagrams are considered in section A.5.2 of 00-58, Part 2. However the guideword interpretations suggested tend towards consideration of design flaws of the diagram (eg: "there is a required relationship that is not shown on the diagram") instead of deviations from design intent of the system described by the diagram. An example of the latter could be: "not all train locations are recorded in the data of the control system" – ie, a discrepancy exists between what the system is supposed to do, according to the diagram, and what it may do in a "part of" deviation.

We suggest the following guideword interpretations for relationships (Table 6).

Guideword	Interpretations
No	No information about this relationship between two objects is recorded by the control system, even though the relation is true in the real world; or the relation does not hold between two real-world objects when it is expected to
More/less	The number of objects in the relationship with another does not obey the cardinality restrictions expected – again this may either be a feature of the real world or erroneous data held by the control system
Part of	Some semantic constraints of the relation given in the diagram hold, but others do not (either in the real world or in the control system data)
Other than	The specified relation does not occur between some objects, another unintended relation is present instead between these objects

Table 6. Guideword Interpretations for Relationships

For classes we could have the interpretations given in Table 7.

For attributes of classes, we could have the interpretations given in Table 8.

1.3 Sequence Diagrams

Sequence diagrams in UML describe the objects and messages involved in a use case (for example, the system reaction to a particular external request or event).

Guideword	Interpretation
No	An object is not a member of an expected class
More/Less	A class has more/fewer instances than expected
Part of	Some of the class constraints are true, others not
Other than	An object is a member of an unintended class

Table 7. Guideword Interpretations for Classes

Guideword	Interpretation
No	An object has no value for this attribute when it should
More/Less	An attribute has a higher/lower value than expected (or more/fewer values than expected, if it is multivalued)
Part of	Some of attribute constraints are true, others not
Other than	Attribute is a member of an incorrect type

Table 8. Guideword Interpretations for Attributes

Compared to class diagrams and statecharts they are imprecise, representing examples or instances of behaviour which could be more comprehensively and accurately defined in these other models. However in the development process sequence diagrams may precede the construction of statechart models, so may therefore be analysed earlier.

The main entities in a sequence diagram are *objects* and *messages*. Objects have the following attributes: (i) their (runtime) class; (ii) their lifetime. Messages have the following attributes: (i) data; (ii) time of occurrence; (iii) source object; (iv) destination object; (v) condition; (vi) delay; (vii) duration (in the case of synchronous messages).

For each attribute of objects and messages in a sequence diagram we can apply guidewords, as follows. Table 9 considers the time of occurrence of a message. For the attribute of *target object* we could have the interpretations given

Guideword	Interpretation
NO/NONE	Message not sent when it should be
OTHER THAN	Message sent at wrong time
AS WELL AS	Message sent at correct time and also an incorrect time
MORE THAN	Message sent later/more often than intended
LESS THAN	Message sent earlier/less often than intended
SOONER	Message sent earlier within message sequence than intended
LATER	Message sent later within message sequence than intended

Table 9. Guideword Interpretations for Message Timing

in Table 10. Table 11 gives interpretations for message conditions.

Guideword	Interpretation
NO/NONE	Message not sent when intended (to any destination)
OTHER THAN	Message sent to wrong object
AS WELL AS	Message sent to correct object and also an incorrect object
REVERSE	Source and destination objects are reversed
MORE THAN	Message sent to more objects than intended
LESS THAN	Message sent to fewer objects than intended

Table 10. Guideword Interpretations for Message Destination

Guideword	Interpretation
NO/NONE	Condition not true when intended
OTHER THAN	Condition has unintended truth value
REVERSE	Condition has opposite truth value to intended
AS WELL AS	An unintended property is true in addition to the condition

Table 11. Guideword Interpretations for Message Conditions

1.4 Def-Stan 00-58 Case Studies and Problems

Annex C of 00-58 part 2 illustrates some PES HAZOPS. These examples are unclear in several cases due to omission of the 'interpretation' column in the tables. This interpretation is still only implicit in the example dialogues and should be made explicit. For example, "Other information is present in the data flow" in C.5.1. Figure 15 has errors (eg, the conditions on the transitions between "Creep" and "low speed" are swapped). In D.4.1 the analysis could be more accurately expressed as a deviation OTHER THAN on the Event attribute of the state machine, ie, an incorrect event is detected by the system. One cause of this incorrect event being detected is wheel spin leading to an incorrect high speed measurement. A consequence is that the system will fail to be activated when it should be (because the real speed is $\leq$ MIN). D.4.2 again concerns the Event attribute on a transition, and guideword NO: failure of the system to register the *vehicle becomes present* event when it should.

1.5 Case Study: Production Cell

An example of application of the above approach is the following, taken from an analysis of a production cell, in which blanks (car body templates) are moved along a feed belt onto an elevating table to be picked up by a robot arm and then transferred to a press for pressing before being transferred to a deposit belt [6]. Figure 1 shows part of the UML class diagram for this system. Applying HAZOPS to this diagram, using table 6, we could have the analyses given in Table 12.

In general either the real world deviates from the model described in the class diagram, and either the software correctly represents this failure, or it does not, or the real world conforms to the model but the software has incorrect

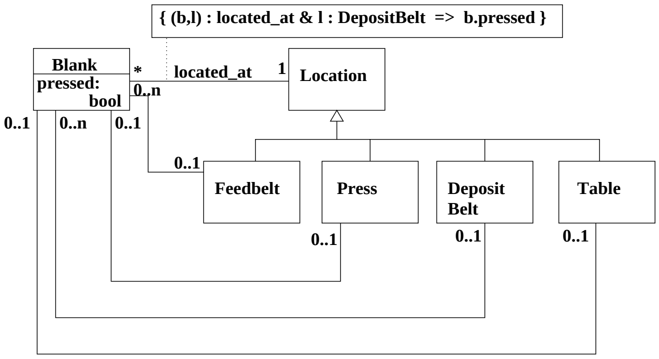

Fig. 1. Analysis Class Diagram for Production Cell

Attribute	Guideword	Interpretations
Blank/table relationship	No	Blank never placed on table after leaving feedbelt. Or system has failed to detect it on table.
	More	2 or more blanks are on the table.
Blank/deposit belt relationship	Part of	There is an unpressed blank on the deposit belt.

Table 12. Guideword Interpretations for Case Study

knowledge. These three cases may be significantly different in terms of deviation causes and consequences, so should be analysed separately.

For instance if a blank has failed to be pressed, and the software fails to detect this, hazards may arise with later processing stages. On the other hand if the blank has been pressed but the software fails to detect this, it may be re-pressed, possibly causing damage to the press.

Figure 2 shows the controller state machine for the feedbelt. $s2$ is the blank sensor at the end of the feed belt, stm a signal indicating that the feed belt is safe to move, and bm is the belt motor. Hazard states are indicated by a cross. HAZOPS of this state machine, using tables 2 and 4, identifies the following potential deviations and hazards (Table 13).

2 Security Analysis

Many safety-critical systems have security issues – eg, in a railway network management system, communication between a train coordinator and train drivers must be authenticated to ensure safe operation of the railway. Other systems may not have direct safety implications (eg, an online banking system) but have security aspects with critical consequences. Traditional HAZOPS concerns deviations from intended functionality, in general. For security HAZOPS we want

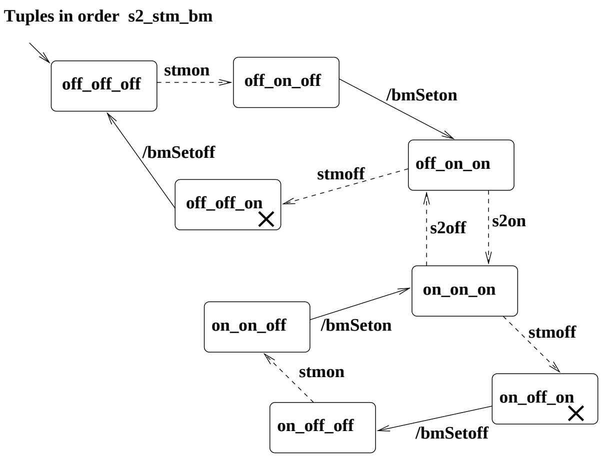

Fig. 2. State machine for Feed belt

Attribute	Guideword	Interpretation
Event (for transition *off_on_on* → *off_off_on*)	No	*stmoff* not detected when it should be. Control system leaves motor on, resulting in hazard state *off_off_on* in EUC.
Action (for transition *on_off_on* → *on_off_off*)	Other than	*bmSeton* issued instead of *bmSetoff*. EUC stays in state *on_off_on* which is hazardous.

Table 13. Guideword Interpretations for Feed belt State machine

to focus on the relevant components (elements which store, process or transmit data internally, or which receive/transmit data to the external world) and relevant attributes, guidewords and interpretations.

Deviation analysis of object-oriented models for security properties can follow the same set of guidewords and interpretations as given above. The elements and attributes of interest will be focussed on security, and similarly for the causes and consequences of deviations. For example in the model of Figure 3 a "part of" deviation could be that some user with a security level below *admin* is logged into the secure host.

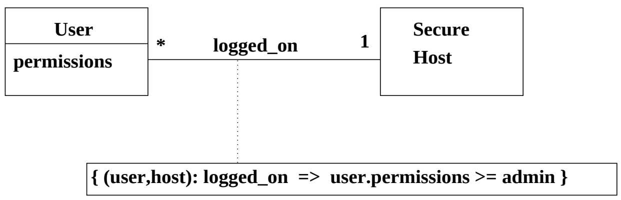

Fig. 3. Part of Class Diagram of Secure System

The aim of security HAZOPS is to identify deviations from design intent which have security implications, ie, which affect the systems ability to:

- ensure confidentiality and integrity of information it stores/processes/internally transmits (*Confidentiality/Integrity*)
- provide authentication of information it produces (outputs) and to authenticate input information (*Authentication*)
- ensure availability of services/information (*Availability*)
- provide a trace of all user interactions, such as login attempts to the system (*Non-repudiation*).

The team approach and other organisational aspects of HAZOPS defined in Def-Stan 00-58 could still be used, although the range of expertise should be relevant (eg, including experts in the encryption/networking approaches used in the system).

The components involved will be:

- Physical components: processing nodes (including intermediate relay nodes in a communication network); memory; connectors; input devices (eg, keyboard; smartcard reader; biometric sensor); output devices (eg, CD burner; smartcard writer; VDU)
- Information components: file, email, network packet. Subcomponents such as particular record fields/email header, etc.

Therefore the guidewords and interpretations could be:

- NO/NONE – complete absence of intended CIA/A/NR property for component being considered

– LESS – inadequate CIA/A/NR property compared with intention
– PART OF – only part of intended CIA/A/NR achieved
– OTHER THAN – intended CIA/A/NR property achieved, but some unintended property also present.

LATE, EARLY, BEFORE, AFTER, INTERRUPT could also be relevant.

The paper [13] gives more focussed guidewords, using the idea of a *guidephrase template*. We can extend this concept to consider the additional properties of authentication and non-repudiation. CIA/A/NR properties may be lost due either to device behaviour (technical failure/cause) or to human actions. Human actions may either be deliberate hostile actions (by insiders or outsiders) or unintentional human failures (or correct procedures which nonetheless lead to a loss of CIA/A/NR). Negations of CIA/A/NR properties are: *disclosure*, *manipulation* (subcases being *fabrication*, *amendment*, *removal*, *addition*, etc), *denial*, *misauthentication* (includes authentication of invalid data and inauthentication of valid data), *repudiation*.

The adapted guidephrase template is given in Table 14. Examples of these

(Pre) Guideword	*Attribute*		*(Post) Guideword*
Deliberate	Disclosure Manipulation	of COMPONENT by	Insider Outsider
Unintentional	Denial Mis-authentication Repudiation		Technical behaviour/ functionality

Table 14. Revised Guidephrase Template

and their interpretations could be:

– *Unintentional misauthentication of connection request to firewall by technical behaviour*. Interpretation (1): firewall authenticates sender of request when it should not. Interpretation (2): firewall does not authenticate a valid sender of request and denies the connection when it should not.
– *Deliberate disclosure of patient record by insider*. Interpretation: staff member provides patient information intentionally to a third party.

We could apply this approach to an example of an internet medical analysis system (Figure 4):

– The company provides a server to perform analysis of data which customers upload to the website, and download results from.
– Data is encrypted by a private key method between the client and server, but stored in decrypted form on the server and the smartcard.
– The smartcard reader at the client computer reads data from the smartcard for transmission to the server.

– Customers have a password/login for the website, this is checked against their customer ID data on the smartcard (written by the server on registration/initialisation of the card). Only if both agree is the login accepted.
– Each customer is allocated a separate directory at the server and cannot load/read the data of any other customer.

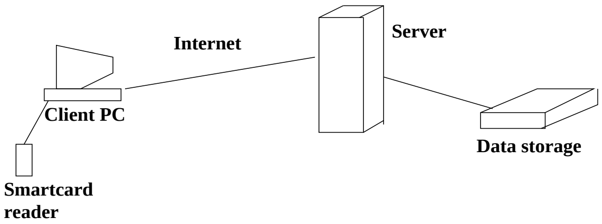

Fig. 4. Architecture of Medical Analysis System

An example security HAZOP analysis from this system is given in Table 15. Detailed risk analysis can then be carried out:

Guidephrase	*Interpretation*	*Causes*	*Consequences*
Unintentional misauthentication of smartcard by technical function	An invalid smartcard is authenticated when it should not be	Software fault in checker, or HW failure in reader	Valid user can use invalid card: may corrupt data stored on server.
Deliberate manipulation of server data by outsider	Server data altered by unauthorised person	Outsider gains login access to server, or via unsecured webpages or scripts.	Analysis data may be altered so incorrect results are given.

Table 15. Example Analysis of Medical System

– Given a specific loss of security situation from the HAZOP, we can compute the approximate likelihoods and severity, and therefore the risk level.
– The probability of decryption of encrypted data can be estimated using mathematical theory, other probabilities, of interception, intrusion, denial of service attacks, are less clear, although estimates based on previous experience could be used.

3 Related Work

State machine hazard analysis has been used informally in the HCI field [2] and tool support for generating fault trees from state machines has been developed as part of RSML [8]. The CORAS Esprit project is investigating security analysis of object-oriented systems, using scenario analysis. Such analysis could be based on the identification of individual deviations using the techniques give here. Tools for HAZOP analysis such as PHA-Pro 5 (http://www.woodhill.co.uk/pha/pha.htm) and PHAWorks 5 (http://www.primatech.com/software/phaworks5.htm) deal mainly with process control system HAZOPS support, and do not cover programmable electronic system designs.

Conclusion

We have described approaches for HAZOPS of UML notations and for security HAZOPS. These revised guidewords and interpretations have been implemented in a hazard analysis tool for UML class diagrams and E/E/PES P&I diagrams within the RSDS support environment [1].

References

1. K. Androutsopoulos, *The RSDS Tool*, Department of Computer Science, King's College, 2001. `http://www.dcs.kcl.ac.uk/pg/kelly/`
2. Alan Dix, Janet Finlay, Gregory Abowd, Russell Beale, *Human-Computer Interaction*, 2nd Edition, Prentice Hall, 1998.
3. ISO, *Guidelines for the Use of the C Language in Vehicle Based Software*, ISO TR/15497. Also at: http://www.misra.org.uk/.
4. J. Julliand, B. Legeard, T. Machicoane, B. Parreaux, B. Tatibouet, *Specification of an Integrated Circuit Card Protocol Application using the B method and linear temporal logic*, B '98, pp. 273–292, Springer-Verlag, LNCS 1393, 1998.
5. J. L. Lanet, A. Requet, *Formal Proof of Smart Card Applets Correctness*, Proceedings of 3rd Smart Card Research and Advanced Application Conference (CARDIS '98), Sept. 1998.
6. K. Lano, D. Clark, K. Androutsopoulos, P. Kan, *Invariant-based Synthesis of Fault-tolerant Systems*, FTRTFT 2000, Pune, India, 2000.
7. P. Lartigue, D. Sabatier, *The use of the B Formal Method for the Design and Validation of the Transaction Mechanism for Smart Card Applications*, Proceedings of FM '99, pp. 348–368, Springer-Verlag, 1999.
8. Nancy G. Leveson. *Designing a Requirements Specification Language for Reactive Systems.* Invited talk, Z User Meeting, 1998, Springer Verlag 1998.
9. Ministry of Defence, Defence Standard 00-56, Issue 2, 1996.
10. Ministry of Defence, Defence Standard 00-58, Issue 2, 2000.
11. Rational Software et al, *OMG Unified Modeling Language Specification Version 1.4*, 2001.
12. K R Leino, J Saxe, R Stata, *Checking Java programs with Guarded Commands*, in *Formal Techniques for Java Programs*, technical report 251, Fernuniversität Hagen, 1999.
13. R. Winther, O-A. Johansen, B. A. Gran, *Security Assessments of Safety Critical Systems using HAZOPS*, Safecomp 2000.