

Component-based HazOp and Fault Tree Analysis in Developing Embedded Real-Time Systems with UML

SHOURONG LU and WOLFGANG A. HALANG
Faculty of Electrical and Computer Engineering
Fernuniversität in Hagen, 58084 Hagen, GERMANY

JANUSZ ZALEWSKI
Department of Computer Science,
Florida Gulf Coast University, Ft. Myers, FL 33965-6565 USA

Abstract: - Hazard and Operability (HazOp) and Fault Tree Analyses (FTA) are embedded into UML component models. The latter are constructed by employing UML's extension mechanisms in conjunction with component-based software techniques. Taking an application's safety-related requirements into consideration, the elements of HazOp and FTA are defined as component attributes, and assigned to a UML component model, which is collected in a UML profile for safety analyses and architectural design. Based on the thus enhanced architecture specification of the UML component model, it becomes possible to handle hazard analyses and to model safety mechanisms at the same time.

Key-Words: - Hazard analysis, software architecture, safety critical system, UML, component, modeling

1 Introduction

Safety analysis ought be on top of the agenda right from start, as it is an integral part of a system's design. It is addressed by a hazard analysis process. Safety analysis teams and engineers produce safety programmes and perform various types of hazard analyses using such techniques as Hazard and Operability Analysis (HazOp), Fault Tree Analysis (FTA), Event Tree Analysis (ETA), and Failure Mode and Effect Analysis (FMEA) [7, 14]. However, these techniques and methods are usually not integrated into the requirements specifications on which software architecture design is being based. This makes it difficult to ensure that an architecture incorporates appropriate safeguards. Architecture is a crucial element in the life-cycle of safety-critical systems, as indicated by the possibility to provide well-known safety mechanisms such as forms of design diversity (e.g., N-version programming) [13]. For these reasons, and to cope with the complexities and safety-related requirements of embedded real-time systems, it may be beneficial to combine in their development process well-established hazard analysis techniques with component-based software engineering which is quite commonly used in traditional applications [12]. A good match appears to be possible on the basis of the semi-formal modeling notation Unified Modeling Language [11].

Unified Modeling Language (UML) offers an unprecedented opportunity to handle both safety analysis and architecture design in the development of safety-

critical systems [2, 4, 5]. The language can be used to construct software architecture specifications dealing with varying levels of modeling abstraction, and to visualise and specify both the static and dynamic aspects of systems [1, 10]. Its built-in extensibility is a powerful feature of UML, providing mechanisms like stereotypes, tagged values and constraints with which the semantics of model elements can be customised and extended. However, safety-related requirements and analyses cannot be described directly in original UML. Therefore, its extensibility needs to be exploited and a UML profile for handling hazard analyses and modeling safety mechanisms is to be built.

To this end, the notations and associated techniques of UML are to be extended in a way consistent with well-defined safety-analysis techniques. In this paper, we shall consider the Hazard and Operability and Fault Tree Analyses, which are known as effective and efficient methods of hazard analysis, and present a component-based way to perform HazOp and FTA on UML models. By employing UML's genuine extension mechanisms in conjunction with component-based software techniques [12], several stereotypes will be defined to incorporate inherently safe elements into the framework of UML, and collected in a profile aiming to address safety analysis and architecture design. The elements in hazard analysis techniques are modeled by UML components, and safety-related parameters are assigned to components as tagged values and constraints. Different

sets of parameters are associated to different kinds of elements.

The paper is organised as follows. Section 2 gives a short overview of the component-based modeling technique in order to provide a basic understanding, and defines a UML component model for developing safety-critical systems, which is a foundation model to embed elements of the hazard analysis techniques. After this, Section 3 explores component-based HazOp and FTA analysis by defining a novel way for architectural specifications with UML notations, and an example are depicted in Section 4. Finally, Section 5 concludes the component-based UML models.

2 A Component-based UML Model

Component modeling deals with three kind of models, namely, structural (static) models, behavioural (dynamic) models, and functional models.

Structural Models consist of components, classes, and their relationships. It represents the static configuration of a system through the dependencies and connections between components. A component can be regarded as a self-contained complex entity consisting of a subcomponent, a class, or a family of subcomponents or classes, common data, and common methods. Each part is linked to its structurally related subcomponents or classes defined by some interaction, which eventually is connected to the outside world via an interface. The component structure is made up of component name, subcomponent name, class name, and interaction name. Components can be subdivided into basic and composite ones: basic components provide simple functionalities, and are executed on hardware devices; composite ones are needed to build hierarchical structures and require other components and connectors.

Behavioural Models are used to describe the dynamic aspects of the components, component interaction and resource constraints. It can be divided into component-interaction parts that show the messages (behavioural name and/or message argument) sent between components (or subcomponents or classes), and state transition parts that present the state transitions of each component (or subcomponent or class) or the interactions between the components (or subcomponents or classes). The component behaviour is constructed with behaviour name, state name, and action name.

Functional Models specify how operations derive output values from input values without regard to the order of computations. Components process inputs according to their operational specification to yield

designated outputs. The component function consists of function name, input parameters, local variables, output parameters, and pre/post-expressions.

The component-based model is defined in UML notations as shown in Fig.1, which is described by defining the set of stereotypes, Component, Connector, Port and Role, Interface and Contract. Component is responsible for the functional aspects, while Connector is responsible for the control and communication aspects of a system. A set of Ports is assigned to Component, and Roles are assigned to Connector. These ports and roles are the interfaces of components and connectors, and obey exactly one protocol which specifies the order of incoming and outgoing messages.

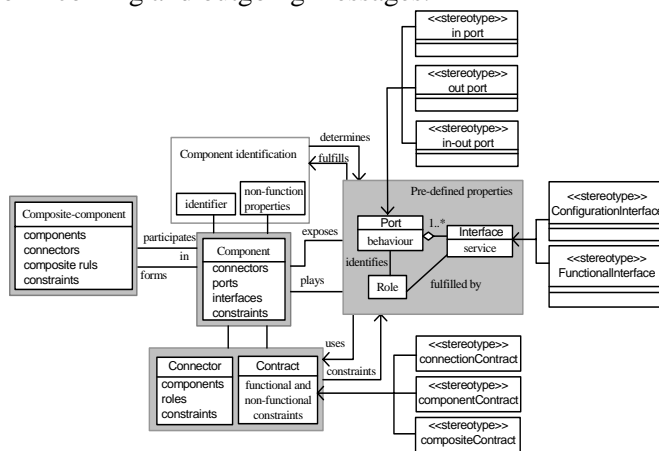


Fig.1 Component-based UML model

The component-based UML model (CBUM) can be formulated by a 7-tuple of the form [3, 8, 9]:

$$CBUM = (n, BSC, CSC, ST, CL, SIG, I)$$

- n : is a unique name of the software component;
- BSC : (BasicComponent) is a set of software components;
- CSC : (CompositeComponent) is a set of software components;
- STC : (StereotypeComponent) is a set of stereotype components;
- CL : (ComponentClass) = $MC \cup CC$ where:
 MC : member class (set of classes)
 CC : common class (set of classes)
 $CC = Met \cup Ath$ where:
 Met : method class (set of classes)
 Ath : attribute class (set of classes)
- SIG is a set of 6-tuple signatures describing operations (functions):
 $SIG = (nf, In, Local, Out, Pre, Post)$ where:
 nf : name of operation (function)
 In : list of input parameters
 $Local$: list of local variables
 Out : list of output parameters
 Pre : expression
 $Post$: expression
- I is a set of 3-tuple signatures describing transactions:
 $I = (Is, Id, Bh)$ where:
 Is : interaction source
 Id : interaction destination
 Bh : is a set of 3-tuple behaviour parts describing the states and actions between Is and Id :
 $Bh = (nb, St, Ac)$ where:
 nb : name of behaviour
 St : set of states
 Ac : set of actions

3 A Hazard Analysis in Component-based Architecture Specification

A hazard analysis serves to identify hazards, determine their respective causes, evaluate the probability of hazards, and to decide on their elimination and mitigation. The causes of a hazard are a set of failures in a system. Thus, its probability can be calculated with the failure probabilities of the architectural elements and the failure probability of their execution behaviour. To treat the hazard analysis aspects we shall enhance architecture specifications with elements of hazard analysis techniques, which serve as basis for construction and evaluation. The elements of hazard analyses are described by UML notations, i.e., the probabilities and failure descriptions are part of the UML component-based model associated to an architectural element. Here we integrate Hazard and Operability Analysis and Fault Tree Analysis into the UML component-based model.

3.1 UML Component-based Model for HazOp

HazOp is a systematic study of how deviations from the design specifications can arise in a system, and whether these deviations can result in hazards [7]. The analysis is performed using a set of guidewords and attributes.

The recommended first step is to identify system entities (elements) and their attributes by examining a description of the system considered. The description of the system may be the one of the physical or logical design. The next step is to apply a number of pre-determined guidewords to attributes of system elements to investigate possible deviations, and to determine the possible causes and consequences of these deviations. A guideword describes a hypothetical deviation from the normally expected attributes. Driven by these guidewords, failure causes and their effects are listed. Each relevant guideword is applied to each attribute, to carry out a thorough search for deviations in a structured manner. The combination of a specific guideword and a particular attribute will need interpretation which may be different in different situations. Each guideword may have more than one interpretation in the context of its application to the design representation. There will be some of the standard guidewords (e.g., *No*, *More of*, *Less of*, *As well as*, *Part of*, *Reverse*, *More than*, *Other than*, etc.) which do not have meaningful interpretations for a particular attribute [6], some systems thus might require the addition of guidewords.

Here we apply HazOp to the UML component-based model specification. By analysing model constructs, identifying attributes of these constructs, and applying guidewords to them, we identify possible deviations and

define analysis criteria for the constructs, i.e., the interpretations of the HazOp guidewords applied to the structural elements composite component, component, connector, interface and port. In the UML component-based model, the model is considered as a HazOp entity, its composed and associated elements can be defined as its HazOp attributes, and the guidewords and their interpretations are defined as *Tag* and *TagValues* applied to these given elements to generate HazOp tables, i.e., the checklists containing suggestions of possible deviations that drives the analysts' attention during the analysis process.

A composite component is constructed by a set of related component instances, which expresses the interactions with one or more internal or external components. A guideword can be used to investigate whether or not an entity contains all of the composed elements necessary to achieve the design intention of the entity within the context of a particular design representation. The component may not only generate failure events, it can also respond (or not) to failure events generated by other components which interface to the component's inputs. Therefore, each of the elements associated with the component can also be considered as an entity during HazOp analysis. Induced by the guidewords of the class Connector, each of the HazOp attributes should be considered vis-à-vis the connections between the components through ports (or interfaces). The guidewords of the component-based model and their interpretations are shown in Table 1.

The state mechanism of UML can be employed to express the dynamic behaviour of components. The structural elements Component, Event, ActiveObject, State, Transition and Activity can be considered as HazOp attributes. The guidewords applied to them are shown in Table 2.

3.2 UML Component-based Model for FTA

The construction of a fault tree provides a systematic method to analyse and document the potential causes of a system failure. The analysis process begins with the failure scenario of interest, and decomposes the failure into its possible causes. Each possible cause is then further refined until the basic causes of the failure are understood. A fault tree consists of several levels connected in such a way that the top event (hazard) is noted at the root of the fault tree. Each event analysed at a given level is connected to its causes (subevents) at the level just below by various Boolean operators. The leaves of the tree are the low-level causes (subevents) for the top event, which have to occur in combination (corresponding to the Boolean conditions in the tree) to trigger the top event. Once a fault tree is constructed, it

Table 1: Guideword interpretations for the component-based model

Attributes (Elements)	Guideword (Tag)	Interpretation (TaggedValues)
Composition component	None	The composite component contains none of the necessary composition components.
	As well As	The composite component contains all necessary composed elements, as well as additional composed elements.
	Part of Other than	The composite component contains some, but not all of the necessary composed components. The composite component contains composed components, but the composed elements do not fulfill the design intention.
Component	None	A subcomponent is not a member of an expected component.
	More / Less	A component has more / fewer subcomponent than expected.
	Part of	Some of the component constraints are true, others are not.
	Other than	A subcomponent is a member of an unintended component.
Connector	No	No connection between two components, or connection does not hold between two components when it is expected to.
	More /less	The specified connector links more / fewer components than expected.
	Part of	Some of the connections are true, others are not.
	Other than	Specified connector does not connect some components, instead, another unintended connection is present between these components.
Port and Interface	No	No connector is represented.
	More	More connectors than intended are represented.
	Less	Less connectors than intended are represented.

Table 2: Guideword interpretations for component behaviour

Attributes (Elements)	Guideword (Tag)	Interpretation (TaggedValues)
Active object	None	The active object is not performed.
	Invalid	The active object is performed in such a way that ill-formed products are produced.
	Incorrect	The active object is performed so that well-formed but incorrect products are produced.
Activity	None	No activity is produced by the active object, or this activity is not transmitted to / carried out by State.
	Invalid	Additional (unwanted) activity is generated / performed, an incomplete activity is generated / performed.
	Incorrect	An incorrect activity is generated / performed.
State	None	Active object not in expected state.
	Other than	Active object in an unexpected state.
Event sequence	Some event lost	Some event does not reach its destination and may affect the interpretation of the subsequent event.
	Events out of sequence	Events, of different types or of the same type, reach their destination in another order than pre-defined.
	Event out of sync	Events are not occurring at the right points in time, missing a time slot or a rendezvous or alike.
	Unknown events	An addressee receives one or many unexpected events.
Event arrival	Too late	The event arrives too late, related to the other operations performed in the unit.
	Never	The event never arrives, possibly blocking operations or causing them to come about differently.
	Unexpected	The event arrives unexpectedly, processed at once or discarded forever.
	Sporadic Too often	The event arrives irregularly, processed at once or discarded forever. The event arrives at too high a frequency, possibly confusing operations or overloading the unit.
Transition	Incomplete	Essential parts of the message are missing.
	Incorrect	The content of the message is incorrect and cannot be interpreted confidently.
	Unchanging	Essential parts of the message are unchanged from one message to the next.

can be written as a Boolean expression showing the specific combinations of identified basic events sufficient to cause the undesired top event.

In order to map the fault tree elements onto the UML component-based model, which enables to use the fault tree to describe the failure behaviours of the component, it is needed to define and describe a *Stereotype* and *Tags* for fault tree elements with UML notations, and to attach them to corresponding elements of the UML component-based model. In the UML component model, a component itself may cause failures such as internal action or internal fault. In addition, it may also respond (or not) to failure events generated by other components which interface to the component's inputs. Therefore, it is natural to define an attribute of failure events in components to express such hazards. These events are defined as stereotypes as shown in Fig.2. Components exchange messages that may cause failures at receiving or sending ports. As in FTA a gate serves as a port endowed by logical functions, we defined a set of gate

stereotypes attaching to ports in UML component models. The FTA construct edge serves as link just as the concept of connector in UML. Hence, we can also define a stereotype Edge, and assign it to Connector. Both new stereotypes Gate and Edge are also shown in Fig.2.













FaultTree Elements	UML Element	Stereotype	Icon
BasicEvent	Component	<<BEvent>>	
UndevelopedEvent	Component	<<UEvent>>	
IntermediateEvent	Component	<<IEvent>>	
NormalEvent	Component	<<NEvent>>	
ConditioningEvent	Component	<<CEvent>>	
AndGate	Port	<<AndGate>>	
ORGate	Port	<<ORGate>>	
InhibitGate	Port	<<InhibitGate>>	
ExclusiveORGate	Port	<<EORGate>>	
PriorityANDGate	Port	<<PANDGate>>	
VotingORGate	Port	<<VORGate>>	
Edge	Connector	<<Edge>>	

Fig.2 Fault tree stereotypes

Stereotype-events represent events such as equipment failures, human errors, software errors, and state occurrences. They are likely to cause undesired outcomes. There are *Basic Events*, i.e., initiating faults requiring no further development; *Undeveloped Events*, i.e., events which are not developed further, either because this is considered unnecessary, or because the information available is insufficient; *Conditioning Events*, i.e., a specific condition or restriction that can apply to some type of Boolean operator; *Normal Event*, i.e., one expected to occur as part of normal system operation; and *Intermediate Event* arising from the combination of other, more basic events.

Stereotype-gates have Boolean functions attached to them. An event is connected to other ones through various Boolean operators (gates). There are several variations and extensions of the connections:

AND gate indicating that all influence factors must apply simultaneously, i.e., a failure in the top node occurs only when all failures in the children nodes occur.

OR gate indicating that at least one of the influence factors must apply to cause the failure, i.e., a failure in the top node occurs only when one or more of the failures in the children nodes occur.

Inhibit gate indicating that a failure in the top node occurs only when both the failures in the child node occur and the condition in the oval is true.

Exclusive OR gate indicating that a failure in the top node occurs only when exactly one of the failures in the children nodes occurs.

Priority AND gate indicating that a failure in the top node occurs only when the failures in the children nodes occur in a left to right order.

Voting OR gate indicating that the output event occurs if a certain number of the input events occur.

Stereotype-edges connect to an event. Edges must not be connected directly to gates or subtree, but only to their input or output ports. Ports can be the sources or targets of edges just as ordinary fault tree nodes (basic events and gates).

4 Example: A Railroad Crossing

Consider a railroad crossing equipped with a semaphore (green/red), which controls the movement of trains, and a gate (up/down), which controls road traffic (Fig.3). Both devices are controlled by a computer system which receives and processes the information related to the train position. The semaphore is red and the gate is up in the initial state of the crossing.

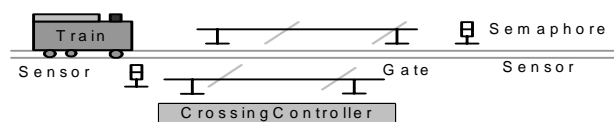


Fig.3 A railroad crossing

To describe the system, a conceptual model needs to be constructed by defining a set of classes and components. The key consideration in constructing the conceptual component model is that it should be a representation of the problem domain, rather than a model of a potential solution [4]. This means that the aim should be to identify the high level concepts within the system and the relationships between them. As shown in Fig.4, the conceptual model is constructed with the crossing controller component that controls train signal actuators and gate actuators. To obtain information from the environment, the control component utilises a sensor that determines the state of the gates, and a sensor that detects an arriving train and its progress through the crossing.

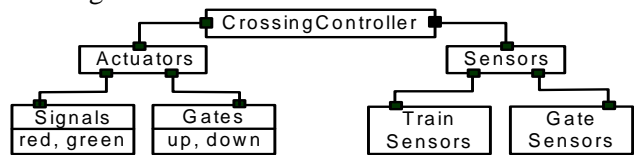


Fig.4 Conceptual model of the railroad crossing

The behaviour of this model is shown as a sequence diagram in Fig.5. When a train approaches a gate, it reaches a given distance where a sensor detects its approach. Then, the sensor sends an event (approach) to the crossing controller which, in turn, causes the actuator with two consecutive commands to close the gate and display green on the semaphore. After the train has passed the gate, a sensor detects the fact and sends an event (pass) to the crossing controller, which issues two commands to display red on the semaphore and to open the gate.

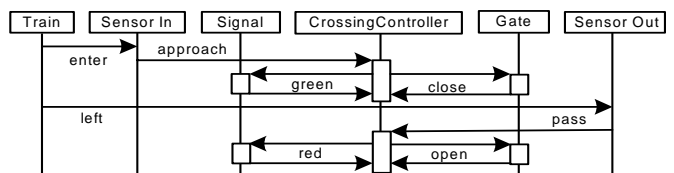


Fig.5 Behaviour at the railroad crossing

After building the component model, the hazard conditions of the system can be discussed. It is obvious that the possible hazards are faulty behaviour of the actuators and sensors, such as the signal green shown to an arriving train in the crossing when the gates are open at the same time. This may be caused by either the system giving a green signal in the wrong situation, or the system omitting to set the signal to red after the train has entered the crossing.

To evaluate the probabilities of undesirable events, an effective way is to construct a fault tree for each component. For this example, the fault trees are constructed with the sensors and actuators as shown in Fig.6. The gates component, for instance, contains an

output failure port for the open and close action, which can be caused either by a failure of the hardware or by a failure of the open gates and close gates action.

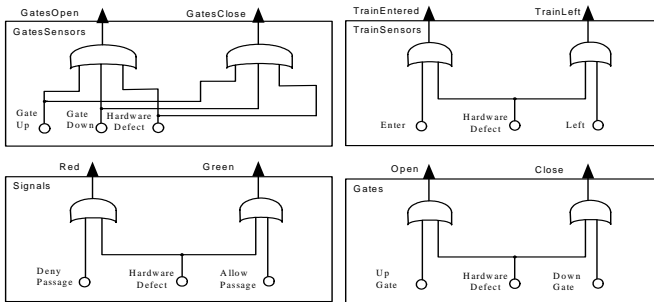


Fig.6 Fault trees of the model's components

Based on each component's fault trees, the composite component fault tree for the railroad crossing control system can be constructed as shown in Fig.7. The probability of a hazard can be evaluated in the same way as for any other fault tree.

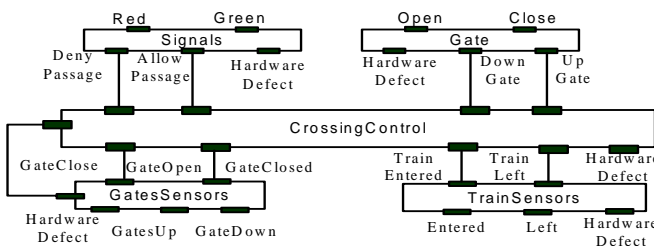


Fig.7 Fault tree of crossing control system

5 Conclusion

The handling of safety analysis techniques within the software architecture has been explored. For this, a set of attributes is defined with UML notations for software architecture specifications of embedded real-time systems. These notations provide the possibility to associate the estimated evaluation properties to the architectural elements. This leads to an improved possibility of finding out whether future systems will meet their safety requirements.

The adaptation of HazOp and FTA to UML component models may provide a useful technique of hazard analysis, which permits qualitative and quantitative analyses at an early design stage, thereby increasing the chances to identify critical behaviour to be analysed and eliminated in the development stages. It helps to systematically identify which hazards and failures are most serious, which components are the most critical ones or which set of components requires a more detailed safety analysis, which consequences of the considered deviations are dangerous as well as how far these deviations depart from the designers' intentions. Furthermore, based on such a safety analysis prevailing Safety Integrity Levels can be derived.

References:

- [1] B.P. Douglass: *Real-time UML: Developing Efficient Objects for Embedded Systems*. Addison-Wesley, 2000.
- [2] H. Giese, M. Tichy and D. Schilling: Compositional Hazard Analysis of UML Component and Deployment Models. *Proc. SAFECOMP 2004*, LNCS 3219, pp. 166--179.
- [3] D. Garlan: Formal Modeling and Analysis of Software Architecture: Components, Connections, and Events. *Proc. 3rd Intl. School on Formal Methods for the Design of Computer, Communication and Software Systems: Software Architecture*, pp. 1--24, 2003.
- [4] R. Hawkins, I. Toyn and I. Bate: An Approach to designing safety critical systems using the Unified Modeling Language. *Proc. Workshop Critical systems development with UML*, 2003.
- [5] B. Kaiser, P. Liggesmeyer and O. Mäckel: A New Component Concept for Fault Trees. *Proc. 8th Australian Workshop on Safety Critical Systems and Software*, 2003.
- [6] S. Kim, J. Clark and J. McDermid: *The Rigorous Generation of Java Mutation Operators Using HazOp*. Technical Report, University of York, 1999.
- [7] N.G. Leveson: *Safeware: System safety and computers*. Addison-Wesley, 1995.
- [8] K.-K. Lau and M. Ornaghi: A Formal Approach to Software Component Specification. *Proc. Specification and Verification of Component-Based Systems*, 2001.
- [9] S. Moschoyiannis and M.W. Shields: Component-Based Design: Towards Guided Composition. *Proc. of Application of Concurrency to System Design*, pp. 122--131, 2003.
- [10] Object Management Group: *Unified Modeling Language: Superstructure*. OMG document ptc/2003-08-02, 2003.
- [11] Object Management Group: *Unified Modeling Language Specification (1.4)*. <http://www.omg.org>, 1999.
- [12] C. Szyperski, D. Gruntz and S. Murer: *Component Software --- Beyond Object-Oriented Programming*. 2nd ed. Addison-Wesley, 2002.
- [13] W. Torres-Pomales: Software Fault Tolerance: A tutorial. *NASA/TM-2000-210616*, 2000.
- [14] J. Zalewski, W. Ehrenberger, F. Saglietti, J. Górski and A. Kornecki: Safety of computer control systems: challenges and results in software development. *Annual Reviews in Control* 27(1) :23--37, 2003.