Developing Safety-Critical Systems with UML

Jan Jürjens*

Software & Systems Engineering, TU Munich, Germany

Abstract. Safety-critical systems have to be developed carefully to prevent loss of life and resources due to system failures. Some of their mechanisms (for example, providing fault-tolerance) can be complicated to design and use correctly in the system context and are thus error-prone. We show how one can use UML for model-based development of safety-critical systems with the aim to increase the quality of the developed systems without an unacceptable increase in cost and time-to-market. Specifically, we describe how to use the UML extension mechanisms to include safety-requirements in a UML model which is then analyzed for satisfaction of the requirements. The approach can thus be used to encapsulate safety engineering knowledge. It is supported by a prototypical XMI-based tool performing the analysis.

1 Introduction

There is an increasing desire to exploit the flexibility of software-based systems in the context of critical systems where predictability is essential. Examples include the use of embedded systems in various application domains, such as fly-by-wire in Avionics, drive-by-wire in Automotive and so on. Given the high safety requirements in such systems (such as a maximum of 10^{-9} failures per hour in the avionics sector), a thorough design method is necessary. In particular, the use of redundancy mechanisms to compensate the faults that occur in any operational system may require complex protocols whose correctness can be non-obvious [Rus94]. Therefore, safety mechanisms cannot be "blindly" inserted into a critical system, but the overall system development must take safety aspects into account. Furthermore, sometimes safety mechanisms cannot be used off-the-shelf, but have to be designed specifically to satisfy given requirements. This can be non-trivial, as spectacular examples for software failures in practice demonstrate (such as the explosive failure of the Ariane 5 rocket in 1997).

Any support to aid safe systems development would thus be useful. In particular, it would be desirable to consider safety aspects already in

* `http://www.jurjens.de/jan` – `juerjens@in.tum.de`

the design phase, before a system is actually implemented, since removing flaws in the design phase saves cost and time. This is significant; for example, in avionics, verification costs represent 50% of the overall costs [Ran00]. There has been a significant amount of successful research into using formal methods for the development of safety-critical systems. Unfortunately, part of the difficulty of critical systems development is that correctness is often in conflict to cost. It would thus be beneficial to use rigorous means in the context of an industrially efficient development method.

The Unified Modeling Language (UML) [UML01] offers an unprecedented opportunity for high-quality critical systems development that is feasible in an industrial context.

- As the *de facto* standard in industrial modeling, a large number of developers is trained in UML, making less training necessary. Also, UML specifications may already be available for safety analysis, which again would save time and cost.
- Compared to previous notations with a user community of comparable size, UML is relatively precisely defined, opening up the possibility for advanced tool-support to assist the development of safety-critical systems.

Problems in critical systems development often arise when the conceptual independence of software from the underlying physical layer turns out to be an unfaithful abstraction (for example in settings such as real-time or more generally safety-critical systems, see [Sel02]). Since UML allows the modeller to describe different views on a system, including the physical layer, it seems promising to try to use UML to address these problems by modeling the interdependencies between the system and its physical environment.

While there has been a significant amount of work addressing real-time systems with UML (including for example [SR98]), and increasing attention to using UML for security (see for example [Jür03c]), in the present work we consider safety and fault-tolerance requirements. To support safe systems development, safety checklists have been proposed in [HJL96, Lut96, Hel98]. Here, we tailor UML to this application domain by precisely defining some such checks with stereotypes capturing safety requirements and related physical properties. This way we encapsulate knowledge on prudent safety engineering and thereby make it available to developers who may not be specialized in safety. One can also check whether the constraints associated with the stereotypes are fulfilled in a given specification. A prototypical tool supporting this will shortly be introduced at the end of this paper.

Safety In safety-critical systems, an important concept also used here is that of a *safety level* (see e.g. [Ran00]). Safety goals for safety-critical systems are often expressed quantitatively via the maximum allowed failure rate. We exemplarily consider the following kinds of failure semantics in this paper (other kinds have to be omitted for space reasons).

- crash/performance failure semantics means that a component may crash or may deliver the requested data only after the specified time limit, but it is assumed to be partially correct.
- value failure semantics means that a component may deliver incorrect values.

Possible failures include:

message loss which may be due to hardware failures or software failures (for example, buffer overflows)
message delay which may in turn result into the reordering of messages if the delay is variable
message corruption when a message is modified in transit.

Forms of redundancy commonly employed include space redundancy (physical copies of a resource), time redundancy (rerunning functions) and information redundancy (error-detecting codes).

UML extension mechanisms We use the three main "lightweight" extension mechanisms (stereotypes, tagged values and constraints) to include safety-requirements in a UML specification, together with the constraints formalizing the requirements. To evaluate a model against the requirements, we refer to a precise semantics for the used fragment of UML extended with a notion of *failures* sketched in Sect. 2.

Related work Some of the ideas reported here were or will be presented in the unpublished [Jür02, Jür03b, Jür03a]. [Jür03c] proposes to use UML for developing security-critical systems. In the related area of real-time systems there has been a substantial amount of work regarding the usage of UML. For example, [SR98] describes constructs to facilitate the design of software architectures in this domain which are specified using UML. [JCF$^+$02] contains several approaches to developing systems with various criticality requirements using UML. In particular, [HG02] discusses a pattern-based approach for using UML use cases for safety-critical systems. The focus is on the development of a testing strategy rather than model analysis. [PMP01] discusses methods and tools for the checking of UML statechart specifications of embedded controllers. The focus is on the use of statecharts and on efficient methods for automated checking and does not include the use of other UML diagrams or the inclusion of safety requirements using stereotypes. Also relevant is the work towards a formal semantics of UML (see the proceedings of related conferences, including the UML and FASE conferences, GROOM UML workshop '98, OOPSLA '98, PSMT '98, ECOOP '00 workshop reader, AMCIS '00, and other conferences). Furthermore, there are techniques for requirements elicitation that can be used fruitfully with the current approach. An example is the approach of goal-oriented requirements engineering [vL01] which is specifically useful for non-functional requirements and which is related to UML in [SC02].

Outline In Sect. 2 we explain the foundation for checking the constraints associated with the stereotypes suggested for safety-critical systems development which are presented in Sect. 3, together with examples of their use. In Sect. 3.1, we shortly describe the tool assisting our approach.

2 Safety Evaluation of UML Diagrams

We briefly give an idea how the constraints used in the UMLsec profile can be checked in a precise and well-defined way. A precise semantics for a (restricted and simplified) fragment of UML supporting these ideas can be found in [Jür03c], building on the statecharts semantics in [BCR00]. It includes activity diagrams, statecharts, sequence diagrams, static structure diagrams, deployment diagrams, and subsystems, each restricted and simplified to keep a mechanical analysis that is necessary for some of the more subtle behavioral safety requirements feasible. The subsystems integrate the information between the different kinds of diagrams and between different parts of the system specification. For safety analysis, the safety-relevant information from the safety-oriented stereotypes is then incorporated as explained below.

Outline of precise semantics In UML the objects or components communicate through messages received in their input queues and released to their output queues. Thus for each component C of a given system, the semantics defines a function $[\![C]\!]()$ which

– takes a multi-set I of input messages and a component state S and
– outputs a set $[\![C]\!](I, S)$ of pairs (O, T) where O is a multi-set of output messages and T the new component state (it is a *set* of pairs because of the non-determinism that may arise)

together with an *initial state* S_0 of the component. The behavioral semantics $[\![D]\!]()$ of a statechart diagram D models the run-to-completion semantics of UML statecharts. As a special case, this gives us the semantics for activity diagrams. Given a sequence diagram S, we define the behavior $[\![S.C]\!]()$ of each contained component C. Subsystems group together diagrams describing different parts of a system: a system component C given by a subsystem $\mathcal{S}$ may contain subcomponents $C_1, \ldots, C_n$. The behavioral interpretation $[\![\mathcal{S}]\!]()$ of $\mathcal{S}$ is defined by iterating the following steps:

(1) It takes a multi-set of input events.
(2) The events are distributed from the input multi-set and the link queues connecting the subcomponents and given as arguments to the functions defining the behavior of the intended recipients in $\mathcal{S}$.
(3) The output messages from these functions are distributed to the link queues of the links connecting the sender of a message to the receiver, or given as the output from $[\![\mathcal{S}]\!]()$ when the receiver is not part of $\mathcal{S}$.

When performing safety analysis, after the last step, the failure model may corrupt the contents of the link queues in a certain way explained below.

Safety analysis For a safety analysis of a given UMLsec subsystem specification $\mathcal{S}$, we need to model potential failure behavior. We model specific types of failures that can corrupt different parts of the system in a specified way, depending on the used redundancy model. For this we assume a function $\mathsf{Failures}_R(s)$ which takes a *redundancy model* R and a stereotype $s \in \{\text{«crash/performance»}, \text{«value»}\}$ and returns a set of expressions $\mathsf{Failures}_R(s) \subseteq \{delay(t) : t \in \mathbb{N} \wedge t > 0\} \cup \{loss(p) : p \in [0,1]\} \cup \{corruption(q) : q \in [0,1]\}$. Here R is a name representing a redundancy mechanism (such as duplication of components together with a voting mechanism), which is semantically defined through the $\mathsf{Failures}()$ sets. The natural number t represents the maximum delay to be expected in time units. p gives the probability that an expected data value is not delivered after the t time units specified in $delay(t)$. Given a value delivered within this time period, q denotes the probability that this value is corrupted.

Then we model the actual behavior of a failure, given a redundancy model R, as a *failure function* that, at each iteration of the system execution, non-deterministically maps the contents of the link queues in $\mathcal{S}$ and a state S to the new contents of the link queues in $\mathcal{S}$ and a new state T as explained below. For this, for any link l, we use a sequence $(lq^l_n)_{n\in\mathbb{N}}$ of multi-sets such that at each iteration of the system, for any n, lq^l_n contains the messages that will be delayed for further n time units. Here lq^l_0 stands for the actual contents of the link queue l. At the beginning of the system execution, all these multi-sets are assumed to be empty. Also, for any execution trace h (that is, a particular sequence of system states and occurring failures describing a possible history of the system execution), we define a sequence $(p^h_n)_{n\in\mathbb{N}}$ of probabilities such that at the nth iteration of the system, the failure considered in the current execution trace happened with probability p^h_n. Thus the probability p^h that a trace h of length n will take place is the product of the values $p^h_1, \ldots, p^h_n$ (since in our presentation here, we assume failures to be mutually independent, to keep the exposition accessible). Then for an execution trace h, the failure function is defined as follows. It is non-deterministic in the sense that for each input, it may have a *set* of possible outputs.

- For any link l stereotyped s where $loss(p) \in \mathsf{Failures}_R(s)$ we
  - either define $lq^l_0 := \emptyset$ and append p to the sequence $(p^h_n)_{n\in\mathbb{N}}$,
  - or append $1-p$ to the sequence $(p^h_n)_{n\in\mathbb{N}}$.
- For any link l stereotyped s where $corruption(q) \in \mathsf{Failures}_R(s)$ we
  - either define $lq^l_0 := \{\Box\}$ and append q to the sequence $(p^h_n)_{n\in\mathbb{N}}$,
  - or append $1-q$ to the sequence $(p^h_n)_{n\in\mathbb{N}}$.
- For any link l stereotyped s where $delay(t) \in \mathsf{Failures}_R(s)$ and $lq^l_0 \neq \emptyset$, we define $lq^l_n := lq^l_0$ for some $n \leq t$ and append $1/t$ to the sequence $(p^h_n)_{n\in\mathbb{N}}$.
- Then for each n, we (simultaneously) define $lq^l_n := lq^l_{n+1}$.

The failure types define which kind of failure may happen to a communication link with a given stereotype, as explained above. Note that for simplicity we assume that delay times are uniformly distributed. Also, corrupted messages (symbolized by $\Box$) are assumed to be recognized (using error-detecting codes). To evaluate the safety of the system with respect to the given type of failure, we define the *execution of the subsystem $\mathcal{S}$ in presence of a redundancy model R* to be the function $[\![\mathcal{S}]\!]_R()$ defined from $[\![\mathcal{S}]\!]()$ by applying the failure function to the link queues as a fourth step in the definition of $[\![\mathcal{S}]\!]()$ as follows:

(4) The failure function is applied to the link queues as detailed above.

Table 1. Stereotypes

Stereotype	Base Class	Tags	Constraints	Description
risk	link, node	failure		risks
crash/ performance	link, node			crash/performance failure semantics
value	link, node			value failure semantics
guarantee	link, node	goal		guarantees
redundancy	dependency, component	model		redundancy model
safe links	subsystem		dependency safety matched by links	enforces safe communication links
safe dependency	subsystem		«call», «send» respect data safety	structural data safety
critical	object	(*level*)		critical object
safe behavior	subsystem		behavior fulfills safety	safe behavior
containment	subsystem		provides containment	containment
error handling	subsystem	error object		handles errors

Containment A system ensures *containment* if there is no unsafe interference between components on different safety levels (this is called *non-interference* in [DS99]). Intuitively, providing containment means that an output should in no way depend on inputs of a lower level. For this we assume an ordering on the set *Levels* of safety levels. Then the *containment* constraint is that in the system, the value of any data element of level l may only be influenced by data of the same or a higher safety level: Write $H(l)$ for the set of messages of level l or higher. Given a sequence $\boldsymbol{m}$ of messages, we write $\boldsymbol{m}\downharpoonright_{H(l)}$ for the sequence of messages derived from those in $\boldsymbol{m}$ by deleting all events the message names of which are not in $H(l)$. For a set M of sequences of messages, we define $M\downharpoonright_H \stackrel{\text{def}}{=} \{\boldsymbol{m}\downharpoonright_H : \boldsymbol{m} \in M\}$.

Definition 1. *Given a component C and a safety level l, we say that C* provides containment *with respect to l if for any two sequences $\boldsymbol{i}, \boldsymbol{j}$ of input messages, $\boldsymbol{i}\downharpoonright_{H(l)} = \boldsymbol{j}\downharpoonright_{H(l)}$ implies $[\![C]\!]\boldsymbol{i}\downharpoonright_{H(l)} = [\![C]\!]\boldsymbol{j}\downharpoonright_{H(l)}$.*

3 Stereotypes for Safety Analysis: The "Safety Checklist"

In Table 1 we give the stereotypes, together with their tags and constraints, that we suggest to be used in the model-based development of safety-critical systems with UML, based on experience in the model-based development of safety-critical systems at our group (for example, from an avionics project [BHL+02]) and on work on safety checklists in the literature [HJL96, Lut96, Hel98]. Thus, in a way, we define a UML-based

Table 2. Tags

Tag	Stereotype	Type	Multipl.	Description
failure	risk	$\mathcal{P}(\{delay(t), loss(p), corruption(q)\})$	*	specifies risks
goal	guarantee	$\mathcal{P}(\{immediate(t), eventual, correct\})$	*	specifies guarantees
model	redundancy	$\{none, majority, fastest\}$	*	redundancy model
error object	error handling	string	1	error object

"Safety Checklist" (which one can verify mechanically on the design level). The constraints, which in the table are only named briefly, are formulated and explained in the remainder of the section. Table 2 gives the corresponding tags. The relations between the elements of the tables are explained below in detail.

Note that some of the concepts introduced below are easier to apply at component rather than object level. Instead of class models, one could also use capsules from [SR98] (but we restrict ourselves to using standard UML 1.5 to remain as general as possible). We explain the stereotypes and tags given in Tables 1 and 2 and give examples (which for space restrictions have to be kept simple). Note that the constraints considered here span a range in sophistication: Some of the constraints are relatively simple (comparable to type-checking in programming languages) and can be enforced at the level of abstract syntax (such as «safe links») and can be used without the semantics sketched in Sect. 2. Others (such as «containment») refer to the semantics and can only be checked reliably using tool-support.

Overview We give an overview of the syntactic extensions together with an informal explanation of their meaning. «redundancy», with associated tag {model}, describes the redundancy model that should be implemented. «risk» describes the risks arising at the physical level using the associated tag {failure}. «guarantee» requires the goals described in the associated tag {goal} for communicated data. «safe links» ensures that safety requirements on the communication are met by the physical layer. «critical» labels critical objects using the associated tags {level} (for each safety level *level*). «safe dependency» ensures that communication dependencies respect safety requirements on the communicated data. «safe behavior» ensures that the system behaves safely as required by «guarantee», in the presence of the specified failure model. «containment» ensures *containment* as defined in Definition 1. «error handling» with tag {error object} provides an object for handling errors.

Table 3. Failure semantics

Risk	$\mathsf{Failures}_{none}()$
Crash/performance	$\{delay(t), loss(p)\}$
Value	$\{corruption(q)\}$

We define the stereotypes and their constraints in detail.

Redundancy The stereotype «redundancy» of dependencies and components and its associated tag {model} can be used to describe the redundancy model that should be implemented for the communication along the dependency or the values computed by the component. Here we consider the redundancy models *none, majority, fastest* meaning that there is no redundancy, there is replication with majority vote, or replication where the fastest result is taken (but of course there are others, which can easily be incorporated in our approach).

Risk, crash/performance, value With the stereotype «risk» on links and nodes in deployment diagrams one can describe the risks arising at these links or nodes, using the associated tag {failure}, which may have any subset of $\{delay(t), loss(p), corruption(q)\}$ as its value. In the case of nodes, these concern the respective communication links connected with the node. Alternatively, one may use the stereotypes «crash/performance» or «value», which describe specific failure semantics (by giving the relevant subset of $\{delay(t), loss(p), corruption(q)\}$): For each redundancy model R, we have a function $\mathsf{Failures}_R(s)$ from a given stereotype $s \in$ {«crash/performance», «value»} to a set of strings $\mathsf{Failures}_R(s) \subseteq \{delay(t), loss(p), corruption(q)\}$.

If there are several such stereotypes relevant to a given link (possibly arising from a node connected to it), the union of the relevant failure sets is considered. This way we can evaluate UML specifications. We make use of this for the constraints of the remaining stereotypes. As an example for a failures function, Table 3 gives the one for the absence of any redundancy mechanism ($R = none$).

Guarantee «call» or «send» dependencies in object or component diagrams stereotyped «guarantee» are supposed to provide the goals described in the associated tag {goal} for the data that is sent along them as arguments or return values of operations or signals. The goals may be any subset of $\{immediate(t), eventual(p), correct(q)\}$. This stereotype is used in the constraints for the stereotypes «safe links» and «safe behavior».

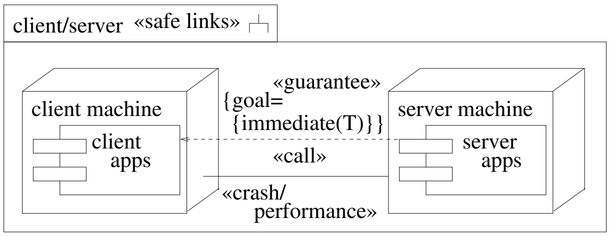

Fig. 1. Example *safe links* usage

Safe links The stereotype «safe links», which may label subsystems, is used to ensure that safety requirements on the communication are met by the physical layer. More precisely, the constraint enforces that for each dependency d with redundancy model R stereotyped «guarantee» between subsystems or objects on different nodes n, m, we have a communication link l between n and m with stereotype s such that

– if {goal} has $immediate(t)$ as one of its values then $delay(t') \in \mathsf{Failures}_R(s)$ entails $t' \leq t$,
– if {goal} includes $eventual(p)$ as one of its values then $loss(p') \in \mathsf{Failures}_R(s)$ entails $p' \leq 1 - p$, and
– if {goal} has $correct(q)$ as one of its values then $corruption(q') \in \mathsf{Failures}_R(s)$ entails $q' \leq 1 - q$.

Example In Fig. 1, given the redundancy model $R = none$, the constraint for the stereotype «safe links» is fulfilled if and only if $T \leq t$, where t is the expected delay according to the $\mathsf{Failures}_{none}(crash/performance)$ scenario in Fig. 3.

Critical We assume that we are given an ordered set *Levels* of *safety levels*. Then the stereotype «critical» labels classes whose instances are critical in some way, as specified by the associated tags {level} (for each level $level \in Levels$), the values of which are data values or attributes of the current object with the required to be protected by the given safety level. This protection is enforced by the constraints of the stereotypes «safe dependency» and «containment» which label subsystems that contain «critical» objects.

Safe dependency The stereotype «safe dependency», used to label subsystems containing object diagrams or static structure diagrams, ensures that the «call» and «send» dependencies between objects or subsystems respect the safety requirements on the data that may be communicated

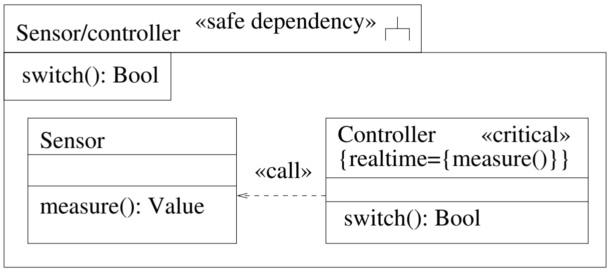

Fig. 2. Example *safe dependency* usage

along them. Exactly, we assume that each $l \in \mathit{Levels}$ has an associated set of goals $\mathit{goals}(l) \subseteq \{\mathit{immediate}(t), \mathit{eventual}(p), \mathit{correct}(q)\}$. Then the constraint enforced by this stereotype is that if there is a 《call》 or 《send》 dependency from an object (or subsystem) C to an object (or subsystem) D then the following conditions are fulfilled.

- For any message name n offered by D, the goals associated with the safety level of n in D imply those in C.
- If a message name offered by D has safety level l in C and $g \in \mathit{goals}(l)$, then g is implied by the goals provided by the dependency.

Example Figure 2 shows a sensor/controller subsystem stereotyped with the requirement 《safe dependency》. We assume that $\mathit{immediate} \in \mathit{goals}(\mathit{realtime})$. The given specification violates the constraint for this stereotype, since Sensor and the 《call》 dependency do not provide the realtime goal *immediate* for measure() required by Controller.

Safe behavior The stereotype 《safe behavior》 ensures that the specified system behavior in the presence of the failure model under consideration does provide the safety goals stated in the tag {goal} associated with the stereotype 《guarantee》 as follows, by referring to the semantics sketched in Sect. 2.

immediate(t) In any trace h of the system, the value is delivered after at most t time steps in transmission from the sender to the receiver along the link l. Technically, the constraint is that after at most t steps the value is assigned to lq_0^l.

eventual(p) In any trace h of the system, the probability that delivered value is lost during transmission is at most $1-p$. Technically, the sum of all p^h for such histories h is at most $1-p$.

correct(q) In any trace h of the system, the probability that delivered value is corrupted during transmission is at most $1-q$. Technically, the sum of all p^h for such histories h is at most $1-q$.

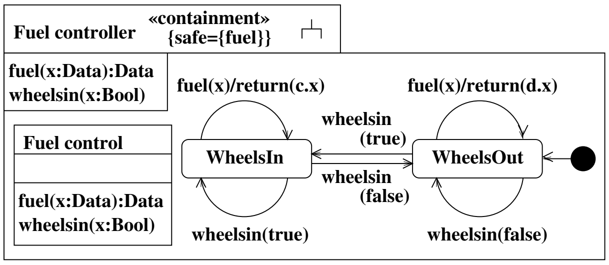

Fig. 3. Example *containment* usage

Containment The stereotype «containment» of subsystems ensures that it satisfies the *containment* constraint defined in Definition 1. For this, we order the set *Levels*: For $l, l' \in \mathit{Levels}$ we have $l \leq l'$ if $\mathit{goals}(l) \subseteq \mathit{goals}(l')$. A subsystem $\mathcal{S}$ correctly carries the stereotype «containment» if $\mathcal{S}$ satisfies containment with respect to every safety level $l \in \mathit{Levels}$.

Example Figure 3 shows a Fuel Controller that computes the amount of used fuel of an airplane from the distance travelled so far. This is done (quite simplistically for the purpose of the example) by multiplying the distance with a constant (the amount of fuel consumed per length unit). Because of different air resistance, this constant depends on the fact whether the wheels of the plane were pulled in-board or (mistakenly) left outside. This is modelled by having two states corresponding to the state of the wheels, and having different constants $c \neq d$. Now the result of the message *fuel* is supposed to be of the level *safe*. However, the message *wheelsin* giving the state of the wheels is not assigned any safety level. Therefore this example violates «containment», because a *safe* value depends on a value not at least of level *safe*. This can be checked as follows: Considering the sequences $\boldsymbol{i} = (\mathit{wheelsin}(\mathit{true}), \mathit{fuel}(1))$ and $\boldsymbol{j} = (\mathit{wheelsin}(\mathit{false}), \mathit{fuel}(1))$, and the safety level $l = \mathit{safe}$, we have $\boldsymbol{i}\restriction_{H(l)} = \boldsymbol{j}\restriction_{H(l)}$, but $[\![p]\!]\boldsymbol{i}\restriction_{H(l)} = \{\mathit{return}(c)\} \neq \{\mathit{return}(d)\} = [\![p]\!]\boldsymbol{j}\restriction_{H(l)}$ since $c \neq d$ by assumption on c, d.

Error handling In UML, an event that does not trigger a transition is ignored. For safety-critical systems the arrival of an unexpected message or of a message with an out-of-range value may indicate a serious failure. Similar to [SP00], we suggest to use a stereotype «error handling» of subsystems with a tag {error object} pointing to a statechart defining the behavior of an error component. This could be compiled directly from the UML model to an aspect-oriented language. For traditional programming languages, we have to "weave" it in on the level of the UML model. For statecharts S_1 defining the component and S_1 defining the error object (both assumed to be flattenable to contain only simple states [LCAK00]) the state set of the result S is the cartesian product of the state sets of S_1 and S_2. There is a transition t from a state (s_1, s_2) to a state (s'_1, s'_2) in S if one of the following conditions holds:

- if there is a transition with the same event, guard, and action from s_1 to s'_1 in S_1 and $s_2 = s'_2$, or
- if there is a transition t' with the same event and action from s_2 to s'_2 in S_2 and the guard of t is the conjunction of the guard of t' and the negation of each guard of a transition t'' in S_1 with the same event, and $s_1 = s'_1$.

3.1 Tool Support

We describe a prototypical tool currently under development for critical systems development for checking constraints such as those associated with the stereotypes defined above. A first version has been demonstrated at [Jür03a]; a web-interface is currently being made available at http://www4.in.tum.de/csduml . The tool works with UML 1.4 models stored in a XMI 1.2 (XML Metadata Interchange) format by a suitable UML design tools. To avoid having to process UML models directly on the XMI level, the MDR (MetaData Repository, http://mdr.netbeans.org) is used, allowing one to operate on the UML model level (this is, for example, used by the UML CASE tool Poseidon, http://www.gentleware.com). The MDR library implements a repository for any model described by a modeling language compliant to the MOF (Meta Object Facility). This approach should easen the transition to future UML versions. To use the tool, a developer creates a model and stores it in the UML 1.4 / XMI 1.2 file format. The tool imports the file into the internal MDR repository and accesses the model through the JMI interfaces generated by the MDR library. The checker parses the model and checks the constraints associated with the stereotype. The results are delivered as a text report for the developer describing problems found, and as a modified UML model, where the stereotypes whose constraints are violated are highlighted. Specifically, the syntactic checks (such as «safe links» and «safe dependency» are implemented in Java, whereas the semantic checks (such as «safe behavior» and «containment») need more specialized tool-support (we are currently working on a connection with Spin and Prolog).

4 Conclusion and Future Work

We propose to use UML to aid the development of safety-critical systems. The goal is to enable developers without a background in safety to make use of safety engineering knowledge encapsulated in a widely used design notation. Since the behavioral parts of UML are supposed to be used with a precisely defined semantics, this allows an evaluation of the models (parts of which may be mechanized). The aim is that the constraints for the safety requirements can be checked and explained to the user by a CASE tool (formally or informally). An XMI-based tool supporting the analysis of the constraints defined here developed in the context of some student projects has been presented at [Jür03a]. It enables developers to analyze UML models created in a UML CASE tool with an XMI output interface for safety and security requirements. As it is still in a prototypical state, interesting further work includes an improvement of the tool to enable use in an industrial environment, which is a necessary prerequisite for technology transfer. Here we only exemplarily realized some of the checks proposed in [HJL96, Lut96, Hel98], for space reasons; others would also be interesting, as well as more specific checks that can lead to safety violations, such as whether variable values are undefined or out of range. Also for space reasons, we could only use very simple examples to demonstrate their use. As [Sel02] points out, dependency of software on the underlying physical layer is not restricted to safety-critical or real-time systems. It would be interesting to see which other application domains beyond safety- (and security-) critical systems our approach could be applied to.

Acknowledgement Progress in tool-support for critical systems development with UML made by P. Shabalin, S. Meng, S. Hoehn, E. Alter and others has contributed to encourage the present work. Comments from the anonymous reviewers improved the presentation of the paper.

References

[BCR00] E. Börger, A. Cavarra, and E. Riccobene. Modeling the dynamics of UML State Machines. In *ASMs*, volume 1912 of *LNCS*. Springer, 2000.

[BHL+02] A. Blotz, F. Huber, H. Lötzbeyer, A. Pretschner, O. Slotosch, and H.-P. Zängerl. Model-based software engineering and Ada: Synergy for the development of safety-critical systems. In *Ada Deutschland 2002*, 2002.

[DS99] B. Dutertre and V. Stavridou. A model of noninterference for integrating mixed-criticality software components. In *DCCA*, San Jose, CA, January 1999.

[Hel98] G. Helmer. Safety checklist for four-variable requirements methods. Technical Report 98-01, Iowa State University Department of Computer Science, 1998.

[HG02] K. Hansen and I. Gullesen. Utilizing UML and patterns for safety critical systems. In Jürjens et al. [JCF+02].

[HJL96] C. Heitmeyer, R. Jeffords, and B. Labaw. Automated consistency checking of requirements specifications. *ACM Trans. on Software Eng. and Methodology*, 5(3):231–261, July 1996.

[JCF+02] J. Jürjens, V. Cengarle, E. Fernandez, B. Rumpe, and R. Sandner, editors. *Critical Systems Development with UML*, number TUM-I0208 in TUM technical report, 2002. UML'02 satellite workshop proceedings.

[Jür02] J. Jürjens. Critical Systems Development with UML. In *Forum on Design Languages*, Marseille, Sept. 24–27 2002. European Electronic Chips & Systems design Initiative (ECSI). Invited talk.

[Jür03a] J. Jürjens. Critical Systems Development with UML, 2003. Series of tutorials at 20 international conferences including SAFECOMP, ETAPS 2003, Formal Methods Europe 2003. http://www4.in.tum.de/~juerjens/csdumltut .

[Jür03b] J. Jürjens. Developing safety- and security-critical systems with UML. In *DARP workshop*, Loughborough, May 7–8 2003. Invited talk.

[Jür03c] J. Jürjens. *Secure Systems Development with UML*. Springer, 2003. In preparation.

[LCAK00] K. Lano, D. Clark, K. Androutsopoulos, and P. Kan. Invariant-based synthesis of fault-tolerant systems. In M. Joseph, editor, *FTRTFT*, volume 1926 of *LNCS*, pages 46–57. Springer, 2000.

[Lut96] R. Lutz. Targeting safety-related errors during software requirements analysis. *The Journal of Systems and Software*, 34:223–230, September 1996.

[PMP01] Z. Pap, I. Majzik, and A. Pataricza. Checking general safety criteria on UML statecharts. In U. Voges, editor, *SAFECOMP 2001*, volume 2187 of *LNCS*, pages 46–55, 2001.

[Ran00] F. Randimbivololona. Orientations in verification engineering of avionics software. In R. Wilhelm, editor, *Informatics – 10 Years Back, 10 Years Ahead*, LNCS, pages 131–137. Springer, 2000.

[Rus94] J. Rushby. Critical system properties: Survey and taxonomy. *Reliability Engineering and System Safety*, 43(2):189–219, 1994.

[SC02] V. Santander and J. Castro. Deriving use cases from organizational modeling. In *RE 2002*, pages 32–42, 2002.

[Sel02] B. Selic. Physical programming: Beyond mere logic. In A. Sangiovanni-Vincentelli and J. Sifakis, editors, *Embedded Software Second International Conference (EMSOFT 2002)*, volume 2491 of *LNCS*, pages 399–406, 2002.

[SP00] P. Stevens and R. Pooley. *Using UML*. Addison-Wesley, 2000.

[SR98] B. Selic and J. Rumbaugh. Using UML for modeling complex real-time systems, 1998.

[UML01] UML Revision Task Force. OMG UML Specification v. 1.4. OMG Document ad/01-09-67. Available at http : //www.omg.org/uml, 2001.

[vL01] Axel van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *RE'01 - 5th IEEE International Symposium on Requirements Engineering*, pages 249–263, Toronto, August 2001. Invited Paper.