Safecharts for Specifying and Designing Safety Critical Systems

Hamdan Dammag and Nimal Nissanke
The University of Reading, Department of Computer Science
Whiteknights, PO.Box 255, Reading RG6 6AY, United Kingdom
Tel: (44) 118 931 8611 (7626), Fax: (44) 118 975 1994
`H.Z.Dammag|N.Nissanke@reading.ac.uk`

Abstract

This paper proposes a novel variant of Statecharts, called Safecharts, especially for use in the specification and the design of safety critical systems. The objective is to provide a sharper focus on safety issues and a systematic approach to deal with them. This is achieved by making a clear separation between functional and safety requirements. A novel feature of Safecharts is the safety annotation, which proposes an explicit ordering of states according to risk level. Transitions are classified according to their risk nature and given a new priority scheme for their execution in the event of any non-determinism. Railway signalling system, a well-known case study, is used as an example to demonstrate some features and semantics of Safecharts.

1 Introduction

The development of critical computer systems invariably requires the use of rigorous techniques in their specification, design and analysis. Statecharts [4] is widely used as a powerful framework for designing real-time reactive systems. Although the Statecharts formalism supports the development of intuitive graphical specifications of such systems, Statecharts as originally proposed have not been adequate for certain tasks. This has led to the proposal of many different Statecharts variants to overcome such inadequacies [1]. Since most of the work related to specifying safety systems is concerned about the change of system states, and their behavioural pattern during the operation of the system, an advantage of Statecharts is that they provide a clear view of the system states and their transformational behaviour.

This paper introduces an extended version of Statecharts, called *Safecharts*, for use in the specification and the design of safety critical systems. The main objective of Safecharts is to provide a sharper focus on safety issues and a framework that forces a disciplined approach to their treatment. The approach makes a clear separation in its representation between functional and safety requirements. The aim is to ensure that safety issues and behaviours of the system are addressed and analysed thoroughly without being distracted by functional design issues.

A typical Safechart representation consists of two separate layers, one dealing with functional behaviour under normal operational conditions and the other dealing with the safety features required under both normal operational conditions and adverse conditions. The two layers are maintained in the form of separate diagrams so that they can be reviewed separately, ideally by different experts. They can be superimposed to form a master diagram, which can be used in examining the interactions between function and safety and the behaviour of the overall system.

In Safecharts, states are ordered according to their risk level. This ordering enables a more meaningful definition of default states, namely as the states of the lowest risk level, referred here by *safe default* states.

Another new feature of Safecharts is the classification of transitions according to the risk level of their target states and the provision of a new priority scheme for their execution. The aim here is to prevent non-determinism becoming a cause for concern from the safety point of view and to resolve it by giving priorities to competing transitions on the basis of safety consideration. The concept of *safe non-determinism* is introduced to represent situations where the non-deterministic modelling of the system has no repercussions on safety. In the safety representation, transitions are annotated with prohibition and mandatory safety clauses in order to constrain transition execution and to ensure the safe evolution of the system.

In order to resolve any resource contention, safety annotation also makes explicit the role of each system component either as a *consumer* or a *resource*. The application of Safecharts is illustrated using Railway Signalling as a case study. Railway signalling has been extensively used in many safety related work [11, 16] and constitutes, therefore, a kind of benchmark for the validation of the Safecharts approach.

2 Statecharts

Statecharts are a visual specification formalism [4, 6] intended for specification and design of reactive systems. Statecharts are a kind of directed graph, with nodes denoting states and arrows denoting labelled transitions. Labels take the form *e[c]/a*, *e* being an event that triggers the transition, *c* a condition that guards the transition when *e* occurs, and *a* an action that is carried out precisely if and when the transition takes place [5]. This allows a tree-like structuring of states, explicit representation of parallelism and communication among parallel components. With the resulting concept of "depth" and superstate-substate relation, the superstate at the top level becomes the specification itself. Statecharts extend conventional FSM's by AND/OR decomposition of states, inter-level transitions and an implicit inter-component broadcast communication.

Once the transition has taken place, the action *a* is generated and broadcast to the whole Statechart, triggering, if applicable, other transitions in the system. Each part of the transition label is optional. That is, if S_2 in Figure 1 is active then the realization of the condition *c* is enough to trigger the transition between S_2 and S_1.

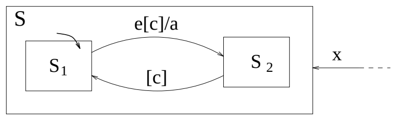


Figure 1: Statecharts transitions between states.

There are three types of states, AND, OR and BASIC. Both OR-state and AND-state consist of a number of substates; being in an OR-state means being in exactly one of its substate, while being in an AND-state means being in all of its substates simultaneously. Substates of an AND-state are indicated by a dashed line and sometimes are called *parallel* states. A BASIC state is the state which has no substates.

For example, the superstate S in Figure 2 is an AND-state which has to be in both substates S_1 and S_2 simultaneously. However, S_1 is an OR-state and, hence, it must be either in C or in D. S_2 is also an OR-state and it must be either in E or in F, where being in F means being in A and B simultaneously.

The default state, pointed by an arrow, is a substate of an OR-state that is to be entered if any transition arriving at the OR-state does not have an explicitly specified entry state. For example, in Figure 2, both D and E are default states for S_1 and S_2 respectively.

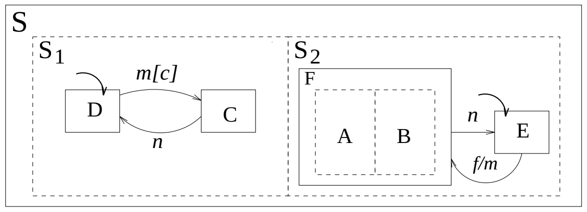


Figure 2: Example of AND/OR decomposition of states.

Orthogonality is a distinctive feature of Statecharts. It is the term used to describe the AND decomposition, where no transition is allowed between the substates of an AND-state. Orthogonality captures *concurrency*; entering an AND-state means entering its every orthogonal component. It also captures *synchronisation* because a single event can cause simultaneous occurrences of several other events. For example, the occurrence of event *n* in Figure 2 causes the transformation from C to D in S_1 and the transformation from F to E in S_2 to occur simultaneously.

The semantics of Statecharts has been introduced in [6]. It has been endowed with a step-semantics in [13], which enforced causality, synchrony hypothesis and global consistency. Despite its success in modelling, the syntax and semantics of Statecharts restricts its usage. In order to overcome these problems, several variants of Statecharts, such as RSML [8], have been proposed. An overview of these variants and their approaches to solve these problems is presented in [1]. For further reading on Statecharts semantics, the reader is referred to [1, 5, 6, 9, 15].

3 Safecharts

Safecharts are an extended version of Statecharts with some unique features. Unlike other Statecharts variants, Safecharts are intended for use exclusively in the specification and design of safety critical systems. The aim is to provide a sharper focus on safety issues of the system. This is achieved by making a clear separation between functional and safety concerns. With this in mind, Safecharts maintain two kinds of behavioural representation:

Functional representation:

This deals with the representation of the states of the system and its components, as well as their transformational behaviour, according to the functionality of the system. This conforms with the conventional use of Statecharts.

Safety representation:

This is an extension to Statecharts and consists of an annotation of safety features and requirements. Other main features of Safecharts are:

1. Special states to identify correct function and malfunction of components and generic events between them to identify component failures and repairs.
2. An ordering of system states according to their relative risk levels. Mathematically, this takes the form of a *risk ordering* relation and is denoted by $\sqsubseteq$.
3. Two kinds of *safety clause*, each giving an additional condition or a constraint on the transition between system states that differ in risk levels: a) a condition inhibiting transitions from low to high risk states, b) a timing constraint in the form of a mandatory deadline on transitions from high to low risk states. The latter kind of transition follows the delivery of a service and is intended as a safety mechanism to avoid imminent failures.
4. Representation of resource dependencies in order to resolve any resource contention. This is achieved by making explicit the role of each system component either as a *consumer*, e.g. an operator request, or a *resource*, e.g. an equipment or a sensor and by defining their interaction.

3.1 Malfunction and repair

Systems and components provide the services expected of them only when they are functioning correctly, but they can fail non-deterministically at any time. The nature of these failure can vary; a detailed discussion may be found in [10]. In order to represent failures, but without distinguishing between different kinds of failures, we introduce for each component two special states, IN and OUT, and two special kinds of generic events, ε and μ. IN represents the component state in which it is functioning normally and OUT represents the state in which it is malfunctioning. An ε event denotes a failure generated internally or by external interference. However, from component's prospective, a failure is always non-deterministic. μ event denotes a maintenance action intended to fix a failure of a system component. Figure 3 shows two possible states of a component; the ε event transforms the component state from the operational state IN into the out-of-order state OUT. The detailed structure of ε and μ events is not of any concern; what is important is their presence or absence. The same applies to the cause and the nature of this failure but, obviously, not to its consequences.

It follows from the broadcasting property of Statecharts that an occurrence of a generic event can affect the internal transitions of the system component. Generic events can be complemented with additional conditions and, furthermore, they can result in actions elsewhere within the system. If it is necessary to identify precisely different failure modes, then the representation can be extended to include several differently labelled OUT states and a range of ε events.

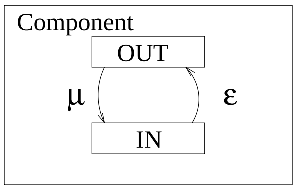


Figure 3: Generic events of a component.

3.2 Ordering of states according to risk

States in Safecharts are ordered according to their relative risk level. In diagrams, higher risk states are placed higher in the Safechart than lower risk states. Mathematically, this takes the form of a *risk ordering* relation on a set of states S. It is denoted by $\sqsubseteq$ such that for two states s_1 and s_2 in S, $s_1 \sqsubseteq s_2$ is true if and only if the risk level of s_1 is known to be less than, or equal to, the risk level of s_2. An underlying assumption here is that risk levels can be quantified. This may be a difficult, or impossible, but we rely on domain experts in defining the relation $\sqsubseteq$. The relation $\sqsubseteq$ may not necessarily be symmetric or antisymmetric, but there can be symmetrical pairs in $\sqsubseteq$, denoting states which are at the same risk level. The relation $\sqsubseteq$ can be represented by means of two relations: a partial order relation $\preccurlyeq$ and an equivalence relation $\approx$ on the set of states S. For any two states s_1 and s_2 in S, $s_1 \preccurlyeq s_2$ is true if and only if the risk level of s_1 is *strictly* lower than of s_2, unless s_1 and s_2 happen to be the same state. Similarly, for any two states s_1 and s_2 in S, $s_1 \approx s_2$ is true if and only if the risk levels of s_1 and s_2 are known, or are assumed, to be identical.

Two states s_1 and s_2 are said to be *comparable* by $\sqsubseteq$ in terms of risk if and only if $s_1 \sqsubseteq s_2$ or $s_2 \sqsubseteq s_1$. that is, their risk levels are known to be identical, or the risk level of one state is lower than that of the other. Otherwise s_1 and s_2 are said to be *non-comparable* through $\sqsubseteq$ in terms of risk. From our point of view, the relation $\sqsubseteq$ is a representation of our knowledge, or the lack of it, about the relative risk levels of states. For example, states (s_1, s_2), (s_3, s_4) and (s_3, s_2) in Figure 4

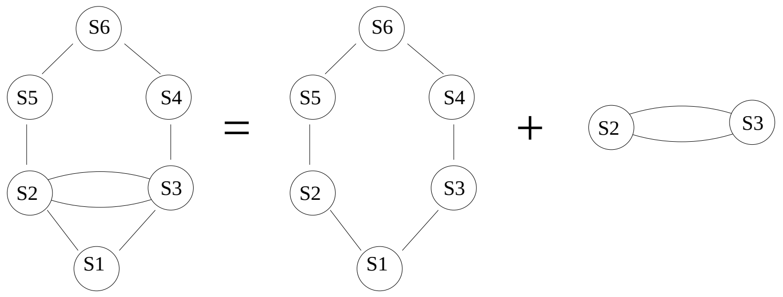


Figure 4: Decomposition of a risk ordering relation.

are comparable in terms of risk, while (s_4, s_5) are non-comparable.

Figure 4 also illustrates the decomposition of a risk graph into a partial order relation $\preccurlyeq$ and an equivalence relation $\approx$. Obviously the partial order relation $\preccurlyeq$ is acyclic and can be defined as the reflexive closure of a precedence relation $\prec$, such that: $\prec \; = \; \sqsubseteq^{sym} \; - \; \sqsubseteq^{-1}$, where $\sqsubseteq^{sym}$ and $\sqsubseteq^{-1}$ are respectively the symmetric closure and the inverse relation of $\sqsubseteq$. In other words, $\preccurlyeq \; = \; \text{id } S \cup \prec$ where id is the identical relation on S. Knowing $\preccurlyeq$, the equivalence relation $\approx$ can be established as: $\approx \; = \; \sqsubseteq \; - \; \preccurlyeq$. Given that $s_1 \prec s_2$, we say that s_1 is of a lower risk level than s_2 or, conversely, that s_2 is of a higher risk level than s_1.

3.3 Diagrammatic Representation

Being devoted to specification and deign of safety critical systems, Safecharts allow separate representation of functional and safety requirements. Separated requirements are represented in the form of two separate layers, one dealing with the functional behaviour under normal operational conditions, and the other dealing with safety features required under both normal operational conditions and adverse conditions. The safety layer facilitates the study of safety concerns of the system without being distracted by the detailed description of its functional behaviour. The two layers can be superimposed to form a master diagram, which can be used in the study of interactions between safety and function and the behaviour of the overall system.

The functional layer conforms fully with the conventional Statecharts and describes the transformational behaviour of the component concerned in providing the service expected of it. However, any transformational behaviour addressing safety requirements does not appear in this layer but in the safety layer of the component.

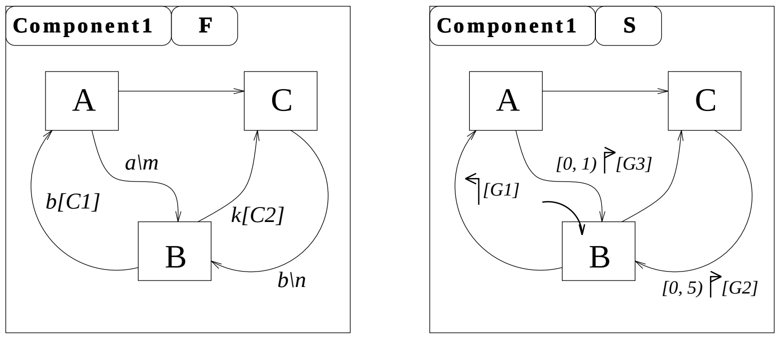
In diagrams, the position of every state in both layers is the same and conforms with its risk level relative to other states, thus, facilitating the integration of different layers. The example in Figure 5 illustrates the separate representation of functional and safety requirements, the layers being labelled as F and S respectively. State A is considered to be of a higher risk level than the state B but in the same risk level with the state C. In other words, $B \preccurlyeq A$ and $A \approx C$.

The ordering of states according to their risk level gives a new meaning to the default state. In Safecharts, default states, referred here as the *safe default* states, are determined according to their risk level rather than their functionality. The safe default state of a component must therefore be one of its safest states. In Figure 5, the state B is the safest state and, therefore, is selected as the safe default state. In the safety layer, transitions are annotated with a safety clause; a detailed discussion follows in the next section.

3.4 Transitions in Safecharts

As in Statecharts, the set of transitions is denoted by T and is defined as a subset of the set $S \times S \times L$, S being the set of states and L a set of labels. Given that $t = (x, y, l)$ is a transition, $source(t) = x$ and $target(t) = y$ denote, respectively, the source and the target states of t. We sometime refer to t also as $x \rightsquigarrow y$. According to the nature of their risk, Safecharts classify transitions into three categories: *safe*, *unsafe* and *neutral*. Safe transition is one whose source state is in higher risk level than its target state. Unsafe transition is one whose source state is in lower risk level than its target state. In neutral transitions the source and target states are at the same risk level.

The above are the only kinds of transition permitted in Safecharts and, thus, exclude transitions between states which are non-comparable in terms of risk. In practical terms, this restriction forces the designer to resolve, as a matter of discipline, the risk levels of any non-comparable states, prior to introducing transitions between them. This is because the presence of non-comparable states is a sign of an incomplete hazard and risk assessment and it is not prudent to introduce



a) Functional layer b) Safety layer

Figure 5: The two layers of Safecharts

any transition between such states without knowing accurately the risks involved in its execution.

Transitions labels in Safecharts have an extended form:

$$e\ [fcond]\ /a\ [l,u)\ \Psi[G]$$

where e is a triggering event, *fcond* is a functional guarding condition, a is an action event, $[l,u)$ is a "right-open" time interval with an inclusive lower bound l and an exclusive upper bound u and $\Psi[G]$ is a safety guarding enforcement on the transition. For convenience, we sometime omit label components when they are irrelevant, or are not essential, to the discussion.

A transition t is said to be *enabled* if $source(t)$ is active, the system continues to remain in that state, the triggering event of t occurs and its functional guarding condition is true. The enabling time of a transition t, denoted by $EnTime(t)$, is defined as the earliest time when all the above become true. The time interval $[l,u)$ is a real-time constraint on a transition t and imposes the condition that t does not execute until at least l time units have elapsed since it most recently became enabled, that is, since $EnTime(t)$, and must execute strictly within at least u time units of $EnTime(t)$. If l and u have not been specified explicitly then the transition is assumed to have an open-ended $[0,\infty)$ time constraint.

The $\Psi[G]$ is a safety enforcement on the transition execution and is determined by the safety clause G. The safety clause G is a predicates which specifies under which conditions a given transition t must, or must not, execute. For each safe or unsafe transition, Ψ is a binary valued constant, signifying one of the following enforcement values:

1. *Prohibition* enforcement value, denoted by $\Lsh$. Given a transition label of the form $t\ \Lsh\ [G]$, it signifies that the transition t is forbidden to execute as long as G holds.
2. *Mandatory* enforcement value, denoted by $\Rsh$. Given a transition label of the form $t\ [l,u)\ \Rsh\ [G]$, it signifies that the transition t is forced to execute within $[l,u)$ whenever G holds.

Two equivelent constructs to the above may be found in [16].

3.5 Resource Dependencies

Safecharts distinguish components as resources and consumers. A resource is represented as parallel OR state with a substate STATUS keeping track of the resource availability. Substates FREE and BUSY of STATUS denote respectively that the resource concerned is available or in use. A consumer acquires free resource by altering STATUS to BUSY and relinquishes it by changing iti back to FREE. The interaction between the consumer and the resources is achieved by using a *state-reference* mechanism. For example, the CONSUMER(1).RESOURCE(A).INS[2].STATUS is a direct reference to the STATUS states of resource instance 2 of type (A) required by the CONSUMER(1). As shown in Figure 6, the general representation of a consumer-resource dependency consists of two main parts: an array of consumers residing in AND-state and a pool of different resources required by the consumers.

In order to avoid any potential resource contention, the consumer does not hold a resource, unless all other resources required by it are also available. This approach makes sure that any potential deadlock situation is prevented since the "Hold and Wait" condition, one of the four necessary conditions for deadlock (see [14] for more details), will not be applicable. Note that it is sufficient for the consumer to know whether or not the required resource is available and unnecessary to know about the consumer, if any, holding it. If there is a conflict between consumers requesting the same available resource at the same time, then one of the pri-

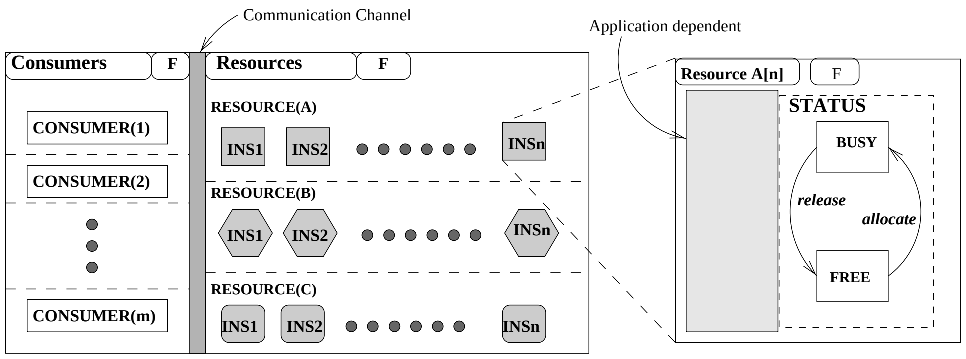


Figure 6: A consumer-resource diagram

ority scheduling algorithms, such as First-Come-First-Served, can be used to select one of the consumers.

3.6 On the Semantics of Safecharts

With respect to some features, Safecharts conform with certain Statecharts variants such as RSML. For example, they allow instantaneous states, parallel execution of transitions and discrete events. Like RSML and other Statecharts variants [12], Safecharts however do not support negated trigger events, but, they have their own semantics with respect to the priority of transition execution.

When modelling safety critical systems, it is important to eliminate any non-deterministic behaviour patterns of the system. However, most Statecharts variants allow the construction of non-deterministic Statecharts. Non-determinism arises if the trigger expressions of two transitions starting from a common state are simultaneously fulfilled. Because of its concern with safety critical systems, Safecharts remove non-determinism in all cases except when there is no safety implication.

It was stated in [1] that the introduction of execution priorities eliminates non-determinism to some extent. In the example given in Figure 7(a), according to the semantics of [6], if the state A is active and the event e occurs then it is not determined whether the transition A$\rightsquigarrow$ B or the transition D$\rightsquigarrow$C is to be executed. However, according to the semantics given in [13] the transition D$\rightsquigarrow$C is the one that should take place since its scope[1] E is on a higher level than the scope D of A$\rightsquigarrow$B .

Nevertheless, the previously mentioned priority concept fails to remove the non-determinism in Statecharts shown in Figure 7(b) since D$\rightsquigarrow$C and A$\rightsquigarrow$B have the same scope, that is the state E. Another priority concept [3] designed to overcome this kind of situations uses the state hierarchy in order to define priorities between transitions. According to this concept, higher the source state of a transition in the hierarchy, higher is its priority. Therefore, in Figure 7(b), it is the transition D$\rightsquigarrow$C which is to be executed if the state A is active and the event e occurs. Even with this priority concept in place, however, all existing Statecharts variants are non-deterministic in the situation illustrated in Figure 7(c).

The ordering of states according to their risk in Safecharts enables not only the determination of safety default states, but also a more sensible approach to assigning priorities to transition execution. Safecharts have different semantics of assigning priorities to competing transitions. Transitions are given priorities according to the risk level of their target states, rather than their scope as in [13], or the hierarchy of their source states as in [3]. For example, since the state C in Figure 7(a) is of lower risk level than state B, D$\rightsquigarrow$C is considered to be a safer transition than A$\rightsquigarrow$B and, therefore, is assigned a higher priority. Therefore, in general, lower the risk level of the target state of a transition, higher the priority of any transition leading to it. Consequently, transition A$\rightsquigarrow$B in Figure 7(c) is the one to be executed if state A is active and the event e occurs. Nevertheless, this semantics can still be non-deterministic, as shown by the example in Figure 7(d). This is because both target states B and C are of the same risk level.

The semantics of Safecharts can be refined further in order to overcome the above situation. This is achieved by considering future transitions, that is, all possible transitions whose source states are those entered by executing all currently competing transitions. This approach is based on future transitions whose source states are of the same risk level but their target states are of different risk levels. For example, in Figure 8(a),

[1]The scope of a transition is the lowest OR-state which is an ancestor state of both its source and target states.

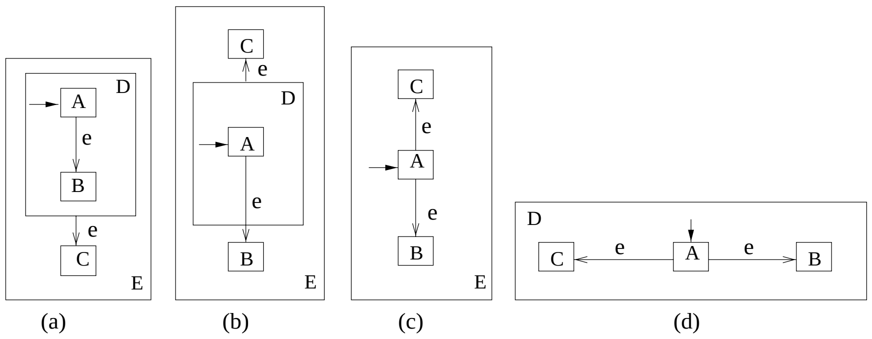


Figure 7: Priorities of transition execution and non-determinism.

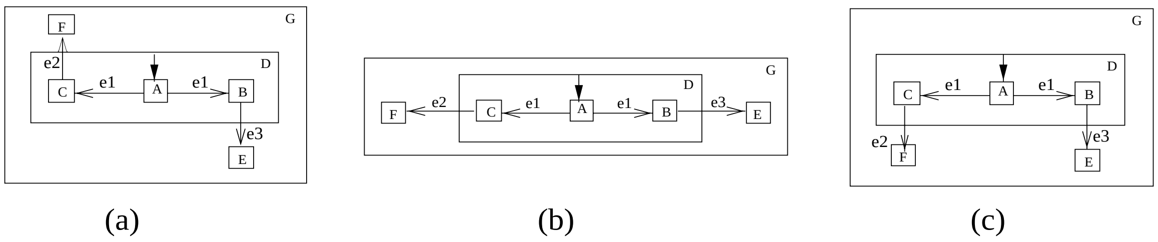


Figure 8: Safecharts approach and safe non-determinism

the transition B⇝E and the transition C⇝F are future transitions whose source states, B and C, are of the same risk level but their target states, E and F, are of different risk level. Since $E \prec F$, the transition B⇝E is safer than the transition C⇝F. Therefore, from the point of view of safety, selecting the transition A⇝B is more appropriate than selecting A⇝C since it will lead to the state B which is a source state of a *safer* future transition B⇝E. An enhancement to the above semantics can be achieved by investigating, if necessary, several steps in order to obtain the safest possible transition among the competing transitions. However, non-determinism can still be encountered if the target states of competing transitions, such as those in Figure 7(d), or the target states of all future transitions, such those in Figure 8(b), are found to be of the same risk level, even if they happen to be at a risk level different from that of the source state of the initial transitions. For example, in Figure 7(d), both C and B are target states of the same risk level and have no future transitions and, therefore, the system will non-deterministically select either A⇝B or A⇝C. In this situation, we are dealing with a series of neutral transitions within the scope of a set of given states, choosing any of them will have no repercussion on safety. For this reason, this kind of situation is referred to here as a *safe non-determinism* of the system.

However, it is worth mentioning that the concept of safe non-determinism includes only the situations where the target states of future transitions are comparable in terms of risk. An example of non-comparable target states (E and F) of transitions is shown in Figure 8(c). Although both transitions B⇝E and C⇝F are safe transitions, the safety of non-detreminism between the transitions A⇝B and A⇝C is not straightforward. This is because it is impossible to determine the safest course of action due to lack of knoweldge about relative safety between F and E. In other words, although both transitions A⇝B and A⇝C will eventually lead to a safer state than the state A, available knoweldge is insufficient to guide the transition execution in the safest possible manner.

We also make an important exception to the rules on priority assignment discussed above. This concerns the generic event ε, introduced in Section 3.1 as a way of modelling component failures. Since failures are inherently non-deterministic and unpredictable, we therefore adopt a principle of preservation of non-determinism with respect to ε events. According to this principle, ε events are always exempt from the priority assignment process.

4 Railway Signalling

This section illustrates Safecharts using railway signalling as a case study. Railway signalling facilitates correct movement of trains, delivering the services in a required manner (function) and not endangering life or property (safety). It consists of two components, namely, *permanent way* (physical equipment: tracks, points and signals) and *interlocking* (safety mechanism that prevents certain undesirable combination of events). Relays on the permanent way detect trains on tracks and sensors monitor the function of signals and points. Both actuators and sensors, as well as the controller itself, can fail at any time, thus, placing users and operators of the railway at risk. For brevity, this paper illustrates Safecharts representations only for signals and routes. However, in the definition of routes (in Section 4.2) we assume the availability of a similar representation for points and tracks.

4.1 Signals

Signals are the standard means for giving instructions to drivers on train movement. Unlike in [11] and [18], signals in our model show only two basic aspects (colours), namely red and green, to represent proceeding and nonproceeding aspects respectively. In order to represent signals in Safecharts, we assume an array of the type: *Signal i : [1..n]*, n being the number of signals in the system. The lamp of a signal can be either operational, indicated by L_IN, or out of order indicated by L_OUT.

The signal changes its aspect from GREEN to RED and, vice versa, when it receives a request from the system. Since signals are considered as resources, as shown in Figure 9, each signal has a state named STATUS. The sensors of the signal, providing the system with information regarding the signal, are also represented. Sensors can either be operational, indicated by S_IN, that is, the system is receiving correct information from the sensor about the signal, or faulty, indicated by S_OUT, meaning the opposite.

If for any reason, the lamp of the signal, or its sensors, fails then the system considers the signal to be faulty, changing it from operational mode to faulty mode. The failure of either of the physical components of the signal is represented by an ε event, while its repair is indicated by μ event, which returns the signal into its operational mode. The actions, which the system may take in response, include withdrawal of the faulty signal from the service and prevention of trains from approaching it.

As part of the safety requirements of the system, in order for the lamp to change its aspect from red to green, the signal ahead must be alight. When the signal returns to its functioning mode, following the repair of its faulty lamp or the sensor, or when it is not required any more by the system, the lamp must show the red aspect. The pointing arrows in Figure 9(b) indicates the safe default states of the signal.

4.2 Routes

A route is defined as a path between two signals in the same direction. Routes are considered as consumer components of the railway signalling system since they consume resource components such as signals, points, and tracks. Each route has a unique identity number. In order to represent routes in Safecharts, we assume the array: *Route* j : $[1..m]$, m being the maximum number of possible routes relevant to the junction (system) under consideration.

If the operator issues a route request over a specific section of the railway then the route request is evaluated by examining the status of the resources required by it, namely, the signals, the points and the tracks involved in the route. In the functional representation of the route, see Figure 10(a), a route request is granted if all resources are available, that is, they are not allocated to another route in the system. If the route was set then all involved resources are consequently set to BUSY. If a second route requests a resource held by the first then it has to wait for the first route to release the resource.

In the safety representation of the route as shown Figure 10(b), it is a safety requirement that for a route to be set, all required resources must be in their operational mode. If any of the resources involved fails after the route has been set then the route is unset and all resources involved in that route are released.

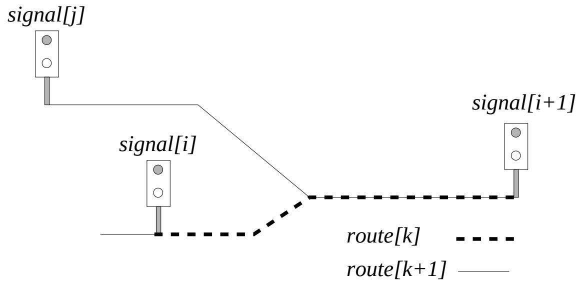
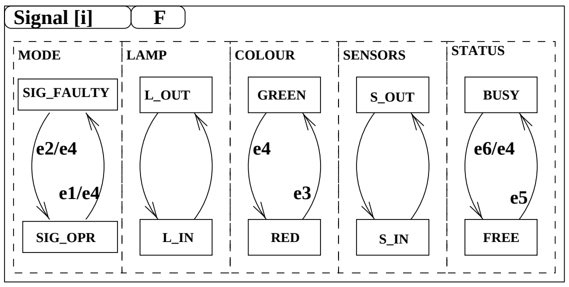


Figure 11: Two routes and a faulty signal.

In order to illustrate how Safecharts work, let us consider two scenarios:

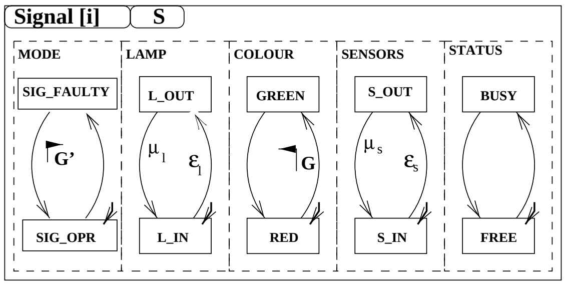
Scenario 1:

When *signal*$[i+1]$ happens to be in a faulty state, that is, in the SIG-FAULTY state, the system requests *signal*$[i]$ to show a proceeding aspect to serve a route request for a train approaching it. The system request is represented by the event e_3, shown in Figure 9(a). However, in the safety layer of *signal*$[i]$, Figure 9(b), the transition RED$\rightsquigarrow$GREEN is constrained by a prohibition enforcement G in the event of *signal*$[i+1]$ being in SIG-FAULTY. Since, in this case, G is true by as-



Symbols	Description	Symbols	Description
e1	ε_l or ε_s	e6	*release*
e2	μ_l or μ_s	ε_l	Lamp failed
e3	*go_green*	ε_s	Sensors failed
e4	*go_red*	μ_l	Lamp repaired
e5	*allocate*	μ_s	Sensors required

a) The functional layer of a signal



Symbols	Description
G	Signal[i+1] in (SIG_FAULTY)
G'	in (L_IN)**and** in (SEN_OPR)

b) The safety layer of a signal

Figure 9: Safecharts representation of a signal.

sumption, then the transition RED⇝GREEN will not be allowed to take place in *signal*$[i]$.

Scenario 2:

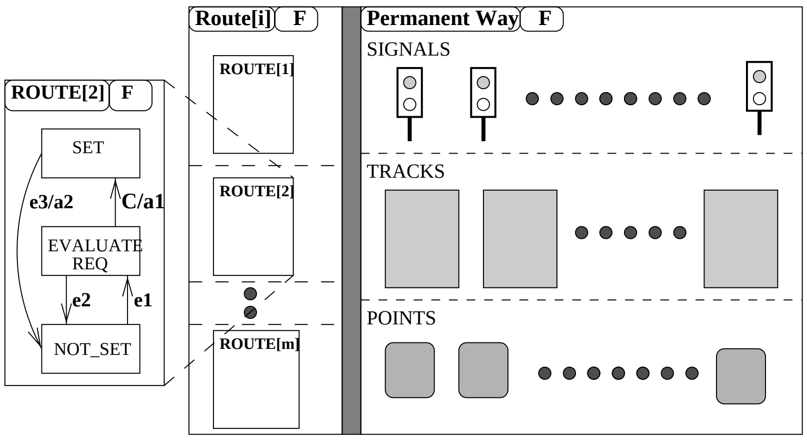
Consider two routes: *route*$[k]$ and *route*[k+1] sharing, as shown in Figure 11, a common resource *signal*$[i+1]$, and that *route*$[k]$ has already been set. Suppose for some reason, the sensor at *signal*$[i]$ fails while the *signal*$[i]$ serving *route*$[k]$. This failure is represented by the generic event ε_s shown in Figure 9(b), which triggers the transition S_IN⇝S_OUT. In this case, event e_1 is generated and causing the transition SIG-OPR⇝SIG-FAULTY to take place; see Figure 9(a). If, at the time when ε_s occurs, the lamp of *signal*$[i]$ is working and showing a proceeding aspect then it must also change its aspect to RED. This is realized by the generation of the event e_4, and the resulting RED aspect ensures that no train passes the given signal. At the same time, since *signal*$[i]$ enters the SIG-FAULTY state, the *route*$[k]$ is unset and forced to release all its resources. Realisation of the mandatory enforcement G attached to the transition SET⇝NOT-SET, in Figure 10(b), forces the transition to take place within a prescribed time limit. Following this, the *route*$[k+1]$ can be set, if necessary, regardless of the faulty *signal*$[i]$, provided that all resources required by it are available.

5 Conclusion

This paper has introduced Safecharts as a novel variant of Statecharts for use in safety critical system specification and design. The strength of Safecharts lies in its systematic approach to design, allowing the designer to first focus on the safety issues without being distracted by its functional requirements and then to consider the interactions between both kinds of requirement. This is achieved by adopting a more appropriate representation for functional and safety requirements.

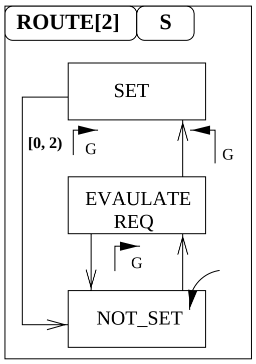
Safecharts has a well-defined semantics expressed in terms of risk ordering relation of the system states and a new meaning of default states. Transitions between states were classified according to the nature of their risk and were given new priority scheme for their execution in the event of any non-determinism. Any non-determinism is resolved in favour of safer transitions that lead the system to a relatively safer state. The term *safe non-determinism* was introduced to express situations where any non-determinism has no repercussion on safety. Transitions were annotated with prohibition and mandatory enforcement as a means of ensuring the safe behaviour of the system. Together these features enhance Statecharts in creating a specialized framework for systematic design of safety critical systems. The approach has been illustrated using railway signalling as a case study.

In order to lessen the effect of exclusion of transitions between all non-comparable states, the current research is considering the concept of broad *bands of risk* so that transitions between non-comparable states belonging to different bands can still be permitted but without affecting the overall safety of the system. Risk bands are numerically ordered from 1 to some n, higher indices implying higher risk. A risk band m contains only comparable states which are related to each other by $\approx$ or non-comparable states whose *immediate* succeding states are in risk band $(m+1)$.



Symbols	Description		Description
e1	*request_route(2)*	Ref1	Route(2).POINTS
e2	*reject_request(2)*	**C**	Ref1 in(FREE) and Ref2 in(FREE) and
e3	*return_route(2)*		Ref3 in(FREE)
Ref1	Route(2).SIGNALS	**a1**	*allocate(Ref1 and Ref2 and Ref3)*
Ref1	Route(2).TRACKS	**a2**	*release(Ref1 and Ref2 and Ref3)*

a) Functional layer of route[2] and its resources.



Symbol	Description
G	Ref1 in(SIG_FAULTY) or Ref3 in(BROKEN) or Ref3 in(DAMAGE)

b) Safety layer of route[2].

Figure 10: Safecharts representation of a route.

References

[1] von der Beek M. *A Comparison of Statecharts variants.* LNCS 863, Springer 1996, Berlin, pp 128-148.

[2] Chan W., Anderson R. J., Beam P., Burns S., Modugno F, Notkin D., Reese J. *Model Checking Large Software Specifications* IEEE, Transaction on Software Engineering, Vol, 24, NO. 7, July 1998.

[3] Day N. *A Model Checker for Statecharts.* Technical Report 93-37, University of British Colombia, Vancouver, Canada, 1993.

[4] Harel D. *Statecharts: A Visual Formalism For Complex Systems.* Science of Computer Programming, North-Holland 1987, Vol. 8, No. 3, pp 231-274.

[5] Harel D., Lachover H., Naamad A., Pnueli A., Politi M. *STATEMATE: A Working Environment for the Development of Complex Reactive Systems.* IEEE Transactions on Software Engineering, Vol. 16, No. 4, April 1994.

[6] Harel D., Schmidt J. P., Sherman R. *On the Formal Semantics of Statecharts.* Proc 2nd IEEE Symposium on Logic in Computer Science, pp 54-64, 1985.

[7] Kesten Y., Pnueli A. *Timed and Hybrid Statecharts and their Textual Representation.* LNCS, VOL. 571, Springer, pp 591-620, 1992.

[8] Leveson N.G., Heimdahl M. *Requirements Specification for Process-Control Systems.* IEEE Transaction on Software Engineering. VOL 20, NO 9. September, 1994.

[9] Maggiolo-Scheltini A., Peron A., Tini S. *Equivalences of Statecharts.* 7th International Conference on Concurrency Theory, Pisa, Italy, 1996.

[10] Nissanke N. *Realtime Systems.* Prentice Hall series in Computer Science, 1997.

[11] Nissanke N., Robinson .N *Formal Methods in safety analysis.* International Conference on Computer Safety, Reliability and Security, Anaheim, California, 1994.

[12] Peron A. *Synchronous and Asynchronous Model for Statecharts* Technical Report TD-21/93, Dipartimento di informatica, Universta di Pisa, Italy 1993.

[13] Pnueli A., Shalev A. *What is in a Step: On the Semantics of Statecharts.* Proc. Symp. on Theor. Aspects of Computer Software, LNCS 526, 244-264, 1991.

[14] Silberschatz A., Galvin P. *Operating Systems Concepts.* Addison-Wesley, 1994.

[15] Sowmya A., Ramesh S. *Extending Statecharts with Temporal Logic.* IEEE Transaction on Software Engineering. VOL 24, NO 3 March, 1998.

[16] Veloudis S., Nissanke N. *Duration Calculus in the Specification of Safety Requirements.* LNCS, Formal Techniques in Real-Time and Fault-Tolerant systems, FTRTFT'98, pp 103-112, 1998.