A Mathematical Framework for Safecharts

Hamdan Dammag and Nimal Nissanke

South Bank University, 103 Borough Road, London SE1 0AA, UK
Email: {*dammagh, nissanke*}*@sbu.ac.uk*
URL: *http://www.sbu.ac.uk/scism*

Abstract. Safecharts is a variant of Statecharts intended exclusively for safety critical systems design. Its specific features include an explicit representation of risks posed by different hazardous states, a separation of functional and safety concerns, a representation of component failures and characterisation of transitions based on the nature of their risk. This paper presents a rigorous mathematical framework for enabling greater clarity and accuracy in Safecharts. It contains a study of the representation chosen for risks and associated concepts such as risk graphs and safety oriented classification of transitions. The step semantics is also defined in relation to Safecharts. As lower level abstractions of states are brought into focus, a way of constructing risk graphs for AND states is suggested. As a case study, the use of Safecharts in the domain of security is illustrated, in particular in modelling the Role-Based Access Control.

Keywords: Safety, Security, Statecharts, Step Semantics, RBAC

1 Introduction

Statecharts, introduced by Harel [6] and extensively studied by others, is widely used for modelling reactive systems. Safecharts [2] is a variant of Statecharts developed originally for exclusive use in the safety critical domain, namely, in relation to systems design. The objectives of this paper are two–fold. On the one hand, the paper aims at providing a mathematical exposition of Safecharts that matches the level of rigour and clarity sought in critical domains such as safety. On the other hand, it attempts to place Safecharts on a new, more general, footing so that it can serve system design with respect to safety, as well as other critical system attributes such as security broadly in a similar manner. This generality is an important goal in itself. It allows the transfer of methodological experience, as well as the associated human expertise, from one domain to another easily, thus mutually enriching the practices of domains concerned in the long run. Despite this generalisation being sought, this paper continues to use, generally, a single 'safety attribute' to mean whatever the critical system attribute under study, whether it is safety, security, or any other system attribute, unless where a reference to a specific system attribute is required.

The basic ideas of Safecharts have been discussed in [2, 12], with illustrations of concepts and usage using case studies. Safecharts aspires to the same goals as Statecharts: visual appeal, ease of abstraction, modular and hierarchical representation of systems, mathematical rigour, etc. In addition, Safecharts aspires to fulfil the needs specific to safety critical systems design; these include representation of risks posed by system states and equipment failures, provision of

additional safeguards against them, a systematic, rigorous and disciplined approach to design, and so on. These attributes are central to the design philosophy adopted in Safecharts.

For achieving a systematic, rigorous and disciplined approach, Safecharts adopts a twin–track strategy. On the one hand, Safecharts uses as its foundation Statecharts – a formalism with an appealing mathematical basis. On the other hand, it separates the aspects of the safety from those of the function, in order to allow the designers to focus on critical and functional features independently and in a systematic manner, and the reviewers to concentrate on safety without being distracted by functional issues. The separation of function and safety is achieved by having two 'layers' in Safecharts representations. The purpose is, on the one hand, to disambiguate between requirements and features devoted to safety and function and, on the other, to highlight the interdependencies between the two. The *functional layer* is devoted to functional issues and utilises Statecharts as used conventionally. The *safety layer* is devoted exclusively to safety issues and deals with issues such as equipment failures, risks posed by hazardous states, representation of safety features and mechanisms and reduction of unpredictable patterns of behaviour due to any non-determinism in a safe manner. In the case of security, safety layer deals with security risks posed by system states and with security mechanisms.

A key feature in realising the above is an ordering of system states according to risks posed by them relative to one another. Mathematically, this corresponds to a *risk ordering relation* on states. As a matter or prudence, Safecharts does not permit transitions between states of unknown risk levels. Recognising the possibility of such a situation arising from omissions, inaccuracies and inconsistencies in the risk ordering relation, for example, due to human error or the lack of knowledge, Safecharts imposes an additional clustering of states into *risk bands* and constructs a *risk graph* of these states. In doing so, any state with a possible inadequate consideration of risk is placed conservatively in a higher risk band by default, alerting the designer to reconsider its risk nature if such an interpretation is undesirable. A classification of transitions into *safe*, *unsafe* and *neutral* transitions based on the risk graph provides a sound basis for calling for additional safeguards against unsafe transitions and prompt enforcement of safe transitions. It also provides a safety-oriented resolution of non-determinism between any conflicting transitions favouring transitions that are more likely to bring the system down to a safer level. Representation of equipment failures and subsequent repair fits in neatly with the proposed framework and allows the incorporation of fail–soft features in functioning equipment in response to failures elsewhere and fail–safe mechanisms in extreme cases.

Correct interpretation of Safecharts requires a sound understanding of several important aspects of its semantics. This paper extends previous work [2, 12] on Safecharts, firstly, by formulating a mathematical framework for dealing with the above issues and, secondly, enriching it with a unique step semantics appropriate to the needs of Safecharts and a set of more refined rules for resolving non-determinism between conflicting transitions. As a new contribution, the papers

makes an advance to the security domain by demonstrating the applicability of Safecharts to modelling of Role Based Access Control [14].

The paper has the following structure. Section 2 introduces basic concepts of Statecharts and the notation used here in relation to Statecharts. Section 3 prepares the ground for the subsequent discussion, introducing the key concepts of Safecharts related only to risks posed by hazardous states. Section 4 presents an integral mathematical view of Safecharts, including its step semantics. Section 5 presents a case study drawn from the domain of security both to illustrate the general use of Safecharts and to point out how it can be used in the security context, while Section 6 concludes the paper.

2 Statecharts

Primary purpose of this section is to place Statecharts in the setting of the mathematical framework used later for defining Safecharts. It is not intended as a formalisation of Statecharts, for which there are widely known other sources. Our formalisation, however, introduces certain restrictions to Statecharts, without greatly inhibiting its generality and yet serving the clarity or simplifications sought in Safecharts. Below is a brief informal introduction to Statecharts.

2.1 Statecharts in Brief

Statecharts is a visual specification formalism introduced by David Harel [6] for modelling the behaviour of complex reactive systems. Statecharts is an extension of finite-state machines with enhanced capabilities such as hierarchical decomposition of systems states, explicit representation of concurrency and broadcast communication. Statecharts is a kind of directed graph, with nodes denoting states and arrows denoting labelled transitions. Labels of transitions take the form $e[c]/a$, e being the triggering event of the transition, c a guarding condition and a an action generated precisely if and when the transition takes place. For a transition to take place, its source state must be an active state. Once generated, the action a is broadcasted to the whole Statechart, triggering, if applicable, other transitions in the diagram. In Statecharts, there are three types of states: AND, OR and BASIC states. Similar to states in state–transition diagrams, BASIC states are non–decomposable. Both AND and OR states consist of a number of substates. Being in an OR state means being in exactly one of its substates while being in an AND state means being in all of its substates simultaneously. The substates of an AND state are indicated by a dashed line and are known as *orthogonal* states.

For example, in Figure 1(a), state S is an AND state with two (orthogonal) substates A and B, each being of type OR. Being in S means being in A and B simultaneously. States D, E, F, G, J and K are BASIC states that cannot be decomposed into further substates. The *default* state, pointed by a dangling arrow, is a substate of an OR state to be entered if a transition arriving at the OR state does not have an explicit entry state. In Figure 1(a), states C and G are the default states of A and B respectively. At initialisation, state S is in its default configuration, namely {J, G}. If the event e occurs, the transition J$\rightsquigarrow$K

takes place. As a consequence, the state J is exited, the state K is entered and the event a is generated and broadcasted throughout the Statecharts. Consequently, the action a triggers transition G⇝F, and hence moving to state F inside state B. As a result, a new configuration of state S is realised, namely {K, F}.

2.2 The Basic Structure of States in Statecharts

As with any *state–based* formalism, fundamental to any definition of semantics of Statecharts are the notions of *state* and *transition*. This approach allows a compositional view of the structure of states at any given level of abstraction, ignoring the internal details of their substates at lower levels of abstraction.

Given an application, let $\mathbb{S}$ denote the set of all relevant states as understood in Statecharts, $\mathbb{T}$ the set of all possible transitions, $\mathbb{E}$ the set of all events, Θ the set of possible types of states in Statecharts, that is, $\Theta = \{\text{OR}, \text{AND}, \text{BASIC}\}$, SN a set of names used for labelling states, and Φ the set of (logical) formulae consisting of variables, logical operators and relational operators. When it stands in for an element of a set, let λ be a *null value*, standing in for an unspecified component, or a component irrelevant to a given specification, belonging to that set. Let S be an arbitrary state in $\mathbb{S}$. It has the general form:

$$S = (id, \theta, C, d, A, \alpha, T, \ell, E) \tag{1}$$

where

- id – $id \in SN$ is a name uniquely identifying S.
- θ – the type of the state S; $\theta \in \Theta$.
- C – a finite set of direct substates of S, referred to as *child states* of S.
- d – $d \in C$ and is referred to as the *default state* of S. It applies only to OR states.
- A – a finite set of currently active child states.
- α – a status flag indicating whether or not S is active; $\alpha \in \{\mathsf{active}, \mathsf{inactive}\}$.
- T – a finite subset of $\mathbb{S} \times \mathbb{S}$, referred to as explicitly *specified transitions* of S.
- ℓ – a function $T \to \mathbb{E} \times \Phi \times \mathbb{F}\, E$, labelling each and every specified transition in T with a triple, $\mathbb{F}\, E$ denoting the set of all finite subsets of E.
- E – the finite set of events relevant to the specified transitions of S; $E \subseteq \mathbb{E}$

When dealing with several states simultaneously, various components of a given state S_i are referred to using the form $S_i.C, S_i.T$, etc. When it makes no confusion, we will denote these components simply as θ, C, etc. Intuitively, an active basic state S has the structure

$$S = (id, \text{BASIC}, \emptyset, \lambda, \emptyset, \mathsf{active}, T, \ell, E) \tag{2}$$

Components in (1) have various interdependencies; their formal definitions and interrelationships are beyond the scope of this paper but are given in [3]. Due to the existence of many different variants of Statecharts (see [15] for a review) transitions have been introduced and interpreted differently. Hence, it is important here to clarify what are valid transitions of Statecharts as understood in Safecharts. Given a transition $t \in \mathbb{T}$, its label is denoted by $\ell(t) = (e, c, a)$, written conventionally as $e[a]/a$. e, c and a in the latter, denoted also as $trg(t) = e$,

$con(t) = c$, and $gen(t) = a$, represent respectively the triggering event, the guarding condition and the set of generated actions. Note that since the elements of the label are optional, these functions may return λ to signify the absence of a particular element of the label. The source state of a transition t is denoted by $sc(t)$ while its target state is denoted by $tg(t)$. When it is more appropriate, a transition will be represented by a pair containing its source and target states, and is indicated as an arrow in the form $sc(t) \rightsquigarrow tg(t)$. For a transition $t \in \mathbb{T}$ to be a *valid* transition, the following conditions must be satisfied: (i) t has only one unique source state and one unique target state. In other words, unlike many statecharts variants, e.g. [7, 9], t cannot have multiple source states or multiple target states, (ii) t does not span between substates of an AND states which is a common ancestor state of its source and target states, and (iii) the source state and the target state of t must not be ancestrally related. For example, according to the above conditions, all transitions in Figure 1(b) are invalid transitions in Safecharts.

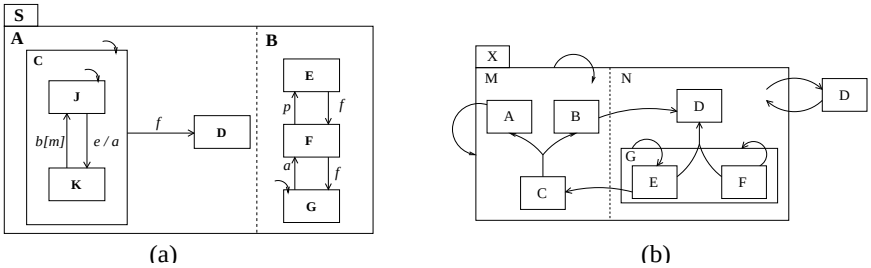

Fig. 1. An Example of Statecharts and invalid transitions.

3 Risk Frame

Turning to the subject matter of this paper, that is, the safety critical systems design, Safecharts treats the hazardous states and the risks posed by them as a fundamentally important issue. In this respect, this section lays the foundation for our subsequent discussion by examining separately several key concepts related to risks posed by states. In the context of a state S, *risk frame* is a 5-tuple and can be defined as:

$$F = (\sqsubseteq, n, \mathcal{B}, \beta, \sqsubseteq_\beta) \tag{3}$$

The key element in F is $\sqsubseteq$, which is a *risk ordering relation* defined on $\mathbb{S} \times \mathbb{S}$. Strictly speaking, $\mathbb{S}$ refers here not to the states in Statecharts but to those in Safecharts. The relation $\sqsubseteq$ expresses risks posed by states in comparison to one another. Other components of the risk frame are actually derived from, or supplement, $\sqsubseteq$ and form the subject of this section. Risk ordering being an outcome of risk assessment, a process conducted by domain experts potentially carrying a degree of human error or misjudgement, the relation $\sqsubseteq$ is prone to gaps, inaccuracies and inconsistencies. *Risk band* is a concept introduced for the purpose of tackling this drawback by placing states conservatively into distinct

bands, or clusters, with some numerical indexing for comparative purposes. n in (3) represents the total number of risk bands of a given state. $\mathcal{B}$ and β are two related functions, the former from $\mathbb{N}$ to $\mathbb{P}\mathbb{S}$ and the latter from $\mathbb{S}$ to $\mathbb{N}$. Given a risk band i and a state s, $\mathcal{B}(i)$ gives the set of states in the ith risk band, while $\beta(s)$ gives the risk band index of the state s . $\sqsubseteq_\beta$ in (3) is a binary relation on $\mathbb{N}_1^k$, with $\mathbb{N}_1 = \mathbb{N} - \{0\}$ and $k = \#S.C$. It is a subsidiary relation for risk ordering in AND states only and is defined using risk band indices of their child states.

3.1 Risk Ordering Relation

Given a state S, its risk ordering relation is denoted by $\sqsubseteq_S$, or simply by $\sqsubseteq$ where it causes no confusion. Given that S is an OR state and the states $s_1, s_2 \in S.C$, the risk ordering relation of S is defined such that $s_1 \sqsubseteq s_2$ 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. The relation $\sqsubseteq$ may consist of pairs of states which are known to be either of two distinct risk levels or of an identical risk level. This can be represented mathematically by decomposing $\sqsubseteq$ into two relations: a partial order relation and an equivalence relation, denoted by $\preccurlyeq$ and $\approx$ respectively. The interpretation of this notation is such that, given two distinct states s_1 and s_2,
$s_1 \preccurlyeq s_2$ – the risk level of s_1 is known to be strictly lower than that of s_2.
$s_1 \approx s_2$ – the risk levels of s_1 and s_2 are known to be identical.
The relation $\sqsubseteq$ is reflexive and transitive. However, $\sqsubseteq$ may not necessarily be symmetric or antisymmetric. This is because there can be symmetrical pairs in $\sqsubseteq$, denoting states which are at the same risk level.

However, in the case of S being an AND state, it is impossible to define the risk ordering relation $\sqsubseteq$ on its parallel child states in C. Alternatively, the risk ordering relation $\sqsubseteq$ is defined on the set C' containing the child states of the equivalent flattened OR state of S, namely S', as mentioned in Section (3.2). The risk ordering relation can be represented as a graph; see Figure 2(a). In order to reduce the clutter in its visual presentations, arcs in graphs $\sqsubseteq$ and $\preccurlyeq$ are assumed to run implicitly upwards and loops at nodes corresponding to reflexive terms are not shown.

3.2 The Risk Graph of OR and AND States

The risk graph of an OR-state S, denoted by $\mathcal{G}(S)$ is constructed on the basis of the risk ordering relation $\sqsubseteq_S$. However, in $\mathcal{G}(S)$, each state is placed in a unique risk band. As an example, Figure 2(b) shows the risk graph of a set of states defined by the relation $\sqsubseteq$ depicted in Figure 2(a). The concept of risk band and its formal definition is given in [3, 12]. Two states s and s' are said to be *risk–comparable* if and only if they are comparable by $\approx$ (i.e. $s \approx s'$) or they lie in different risk bands. Otherwise, they are said to be *risk–noncomparable*. Note that risk–comparable states in the risk graph may be noncomparable by the relation $\sqsubseteq$.
In general, risk ordering of an AND state can be quite complicated and may not be a viable option in practice when dealing potentially with a large number of orthogonal child states. This is because risk ordering in an individual orthogonal

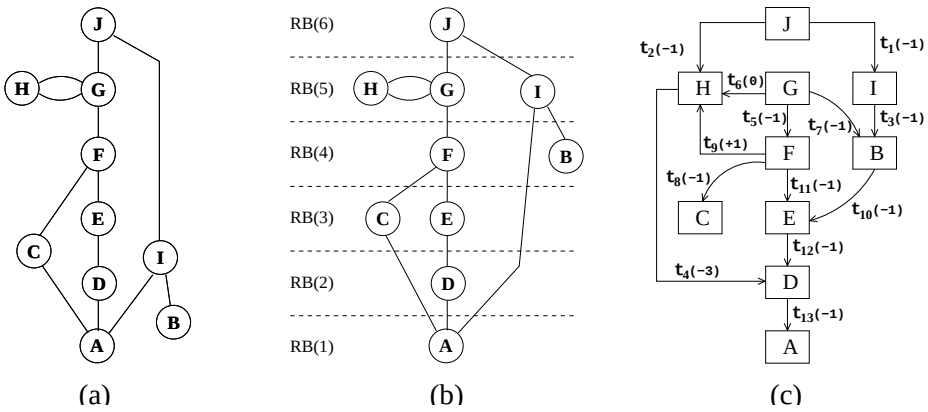

Fig. 2. (a) Risk ordering relation (b) Risk graph (c) Risk distances of transitions.

child OR state would no longer make sense unless due attention is paid to risk ordering in the adjoining child OR states. However, this difficulty can be overcome by *flattening* the AND state, that is, by converting the AND state into its equivalent OR state. In doing so, the risk ordering relation can be applied to the resulting OR state and, hence, the risk graph can be constructed in the usual manner.

An AND state S, with a set of direct substates C, can be flattened into an equivalent OR state S' whose C' consists of tuples drawn from the unordered Cartesian product of all orthogonal states in C. Each such tuple consists of a number of *parallel* states, equal to the number of orthogonal states in C and corresponds to a conventional state. The transitions associated with the equivalent OR state can be derived using the canonical mapping approach of [5]. For example, Figure 3(a) shows an AND state with two orthogonal substates M and N, while Figure 3(b) shows the equivalent OR state as well as its interpreted transitions.

When flattening an AND state, the number of interpreted transitions rises rapidly, especially if the AND state consists of many orthogonal states. This is a well–known problem of state–transitions diagrams – a problem that led to the very invention of Statecharts in the first place. Statecharts achieved this through the notions of depth and abstraction. In this context, flattening AND state amounts to the reverse process but is necessitated by the need to consider the risks posed by possible combinations of states – a requirement peculiar to critical systems.

Depending on the performance of the domain expert, the risk graph of an AND state can be specified either: (i) *directly*, or (ii) *indirectly*. In (i), the risk ordering relation $\sqsubseteq$ can be applied to the OR state obtained by flattening the AND state. Hence, the risk graph can be constructed in the usual manner. In (ii), risk ordering can be done using an irreflexive subsidiary risk ordering relation $\sqsubseteq_\beta$ defined in terms of the risk band indices of individual orthogonal risk

graphs. Thus, (ii) does not require flattening the AND state for the purpose of specification of $\sqsubseteq$. A formal definition of the direct and indirect approaches is given in [3].

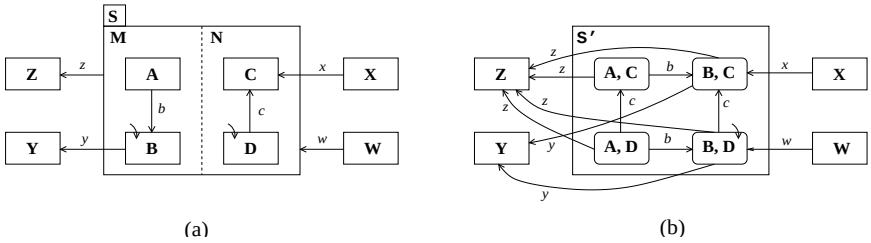

Fig. 3. An AND state and its equivalent flattened OR state.

3.3 Node Replacement in the Risk Graph

The hierarchical structure of states in Statecharts, achieved by the AND and OR composition, is also reflected in risk graphs. Analogous to a state in Statecharts being composed of a number of other child states, a node in a risk graph may in turn consist of a risk graph corresponding to the structure of the state represented by that node. When dealing with the system under consideration at a lower level of abstraction, there may be a need to expand the risk graph to the same level of abstraction. In this case, it is necessary to replace the node concerned with the risk graph it represents. This section outlines how to perform such node replacement. Given that x is a non BASIC state and $x \in S.C$, the node corresponding to x in $\mathcal{G}(S)$ can be replaced by its risk graph $\mathcal{G}(x)$ in the following manner. Let $\mathcal{G}'(S)$ denote the revised risk graph of S after the node replacement. Nodes in the highest risk band of a risk graph are referred to as its 'highest nodes' while its 'lowest nodes' are those nodes in the lowest risk band:

(a) The node x, as well as arcs incident on it, are removed from $\mathcal{G}(S)$.
(b) The highest node(s) in $\mathcal{G}(x)$ are connected to immediate successor nodes of x in $\mathcal{G}(S)$, if any.
(c) The lowest node(s) in $\mathcal{G}(x)$ are connected to immediate predecessor nodes of x in $\mathcal{G}(S)$, if any.
(d) If there exists a node representing a state x' in $\mathcal{G}(S)$ such that $x' \approx x$
  (i) if x' is a BASIC state then the lowest node(s) of $\mathcal{G}(x)$ are connected to x' by the $\approx$ relation,
  (ii) if x' is a non-BASIC state then the lowest node(s) of $\mathcal{G}(x)$ are connected to the lowest node(s) of x', by the $\approx$ relation.
(e) In the event of x having no direct successor in $\mathcal{G}(S)$ but there being a node x' in $\mathcal{G}(S)$ such that $\beta_S(x') = \beta_S(x)+1$ in $\mathcal{G}(S)$, highest node(s) of $\mathcal{G}(x)$ are to be placed in $\mathcal{G}'(S)$ at least one risk band lower than that of x' in $\mathcal{G}'(S)$.
(f) In the event of x having no direct predecessor in $\mathcal{G}(S)$ but there being a node x' in $\mathcal{G}(S)$ such that $\beta_S(x') = \beta_S(x)-1$ in $\mathcal{G}(S)$, lowest node(s) of $\mathcal{G}(x)$ are to be placed in $\mathcal{G}'(S)$ at least one risk band higher than that of x' in $\mathcal{G}'(S)$.

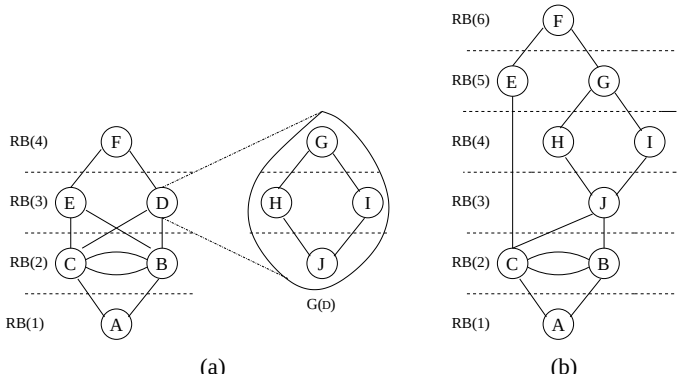

Fig. 4. Replacing node D with its risk graph $\mathcal{G}(\text{D})$

The risk graph $\mathcal{G}(x)$ intended to replace the node x is obtained depending on the nature of the node x such that: (i) if x denotes an AND or an OR state then $\mathcal{G}(x)$ is obtained as described in Section (3.2), or (ii) if x denotes a tuple then $\mathcal{G}(x)$ is obtained analogous to the risk graph of an AND state. This is achieved by considering x as being an AND state and the elements of the tuple as being its orthogonal states. However, in contrast to our definition of AND states, in this case, the set $x.C$ (the elements in the tuple) might contain one, or more BASIC state(s). In this case, the risk graph $\mathcal{G}(x)$, is constructed as follows: (a) all BASIC states in the tuple x are to be excluded and a risk graph $\mathcal{G}(x')$ of the remaining non-BASIC states is to be conventionally constructed, and (b) every excluded BASIC states is to be attached to every node in $\mathcal{G}(x')$. In the case where all the elements in x are BASIC states then no node replacement takes place.

In the interest of maintaining an identical level of abstraction, it makes sense to perform any node replacement on all non-BASIC nodes at a given level of the state hierarchy simultaneously. This is in line with maintaining the degree of the depth and abstraction obtained by the Safecharts diagram. As an example, Figure 4(a) shows the risk graph of S with node D to be replaced by its risk graph, that is by $\mathcal{G}(\text{D})$. The revised risk graph of S, that is $\mathcal{G}'(S)$, obtained after the node replacement is shown in Figure 4(b).

4 Safecharts

Safecharts was introduced in [2] as a safety-oriented variant of Statecharts developed especially for the specification and design of safety-critical systems. One of its unique features is the maintenance of two separate layers of representation: a *functional layer* and a *safety layer*. The aim of the former is to capture system's transformational behaviour purely from a functional point of view, by using Statecharts in the conventional sense, while that of the latter is to capture the risk involved in such behaviour. The safety layer contains a *risk graph* of the states of the system under description and a *safety annotation* associated

with transitions between these states. The concept of *risk graph* is based on our discussion in Section 3.

4.1 States in Safecharts

In dealing with failures in safety–critical systems, each component is represented in the form of an OR state with two distinguished substates, denoted generically by IN and OUT, meaning respectively that the component is functioning correctly or has failed. The nature of these two states are such that IN is strictly safer than OUT (IN $\preccurlyeq$ OUT). Associated with these states are also two generic events: a non-deterministic event ε signifying a failure, and an event μ signifying a maintenance or repair action which returns the component back to service. A component may have more than one failure mode, in which case OUT may itself be an OR state with a distinct substate for each of the failure modes, possibly with further transitions to model failure propagation.

Let us refer to the notation introduced in Section 2.2 in relation to **St**ate**c**harts using the subscript *stc*, and to the notation introduced here in relation to **S**afe**fc**harts using the subscript *sfc*. First let us define a predicate *sys* on $\mathbb{S}_{stc}$ such that $sys(S)$ is true of S exactly if it models the state of a system, or the state of an item of equipment. For each such state in $\mathbb{S}_{stc}$, let there be a corresponding state $\mathbb{S}_{sfc}$ with three further states SYS_S, IN_S and OUT_S. Informally, SYS_S denotes the state of an extended *failure-prone* version of S. The $\mathbb{S}_{sfc}$, and likewise the $\mathbb{E}_{sfc}$, can be extended as follows:

$$\mathbb{S}_{sfc} = \mathbb{S}_{stc} \cup \{\text{SYS}_S, \text{IN}_S, \text{OUT}_S \mid S \in \mathbb{S}_{stc} \wedge sys(S)\} \tag{4}$$

$$\mathbb{E}_{sfc} = \mathbb{E}_{stc} \cup \{\epsilon_S, \mu_S \mid S \in \mathbb{S}_{stc} \wedge sys(S)\} \tag{5}$$

where $\text{IN}_S, \text{OUT}_s, \epsilon_S$ and μ_S are as introduced above. Where it causes no confusion, the subscripts in these new elements will be dropped. The states in Safecharts have an extended structure and include, in addition to what was discussed in section 2.2, the components of the risk frame as well as the associated relations $\sqsubseteq$ and $\approx$. The extended structure has the form

$$S = (id, \text{OR}, C, d, A, \alpha, T, \ell_s, E_s, \sqsubseteq, n, \mathcal{B}, \beta, \lambda) \tag{6}$$

$$S = (id, \text{AND}, C, \lambda, A, \alpha, T, \ell_s, E_s, \sqsubseteq\ n, \mathcal{B}, \beta, \sqsubseteq_\beta) \tag{7}$$

$$S = (id, \text{BASIC}, \emptyset, \lambda, \emptyset, \alpha, T, \ell_S, E_S, \lambda, \lambda, \lambda, \lambda, \lambda) \tag{8}$$

where $E_s \in \mathbb{E}_{sfc}$ and every component is defined using the sets (4) and (5) with extended versions defined above. When dealing with the two special states IN and OUT, let us separate the functional and safety requirements concerning the state S as follows:

$$\text{IN} = S_f {+\!+} S_s \tag{9}$$

$$S_f = (id, \theta, C, \lambda, A, \alpha, T_f, \ell_f, \mathbb{E}_{stc}, \lambda, \lambda, \lambda, \lambda) \tag{10}$$

$$S_s = (id, \theta, \lambda, d, \lambda, \alpha, T_s, \ell_s, \mathbb{E}_{sfc}, \lambda, \sqsubseteq, n, \beta, \sqsubseteq_\beta) \tag{11}$$

S_f and S_s being two partially completed templates of state specifications in Safecharts. The operator $+\!+$ which 'glues' the two templates together, is intended to have the following effect: $\theta_{\text{IN}} = \theta_{S_f} = \theta_{S_s}$, $C_{\text{IN}} = C_{S_f}$, $d_{\text{IN}} = d_{S_s}$,

$A_{\text{IN}} = A_{S_f}$, $T_{\text{IN}} = T_{S_f} \cup T_{S_s}$, $E_{\text{IN}} = E_f \cup E_s$ and $\ell_{\text{IN}} = \{(t, \ell_f(t) \frown \ell_s(t)) \mid t \in T_{\text{IN}}\}$. The risk frame components in state IN are identical to those in S_s. If necessary, state OUT may also be defined as an OR state for modelling failure propagation from one mode to another. The extended failure–prone version SYS for a given state $S \in \mathbb{S}_{stc}$ is an OR state that can be defined as follows:

$$\begin{aligned}\text{SYS} = (id, \text{OR}, \{\text{IN}, \text{OUT}\}, \text{IN}, A, \alpha, \{(\text{IN}, \text{OUT}), (\text{OUT}, \text{IN})\},\\ \ell_s, \{\epsilon, \mu\}, \{(\text{IN} \prec \text{OUT})\}, 2, \{(\text{IN}, 1), (\text{OUT}, 2)\}, \lambda\})\end{aligned} \tag{12}$$

Features SYS (12), IN (9) and S_s (11) represent the contents of the safety layer, whereas S_f (10) represents the content of the functional layer. It is clear that S_f in (10) is based purely on Statecharts as understood conventionally. Thus, its non-null values are identical to the corresponding ones given in (1) for dealing with functional requirements. The default state d in S_s (11) is defined such that $\beta(d) = 1$ and is referred to as the *safe default* state of IN, which is itself being the default state of SYS (12). This forms a safe initialisation feature in Safecharts.

4.2 Transitions in Safecharts

A transition $t \in T$ in Safecharts is a *legal* transition if and only if $sc(t)$ and $tg(t)$ are risk-comparable states in a common risk graph. Based on the risk graph, Safecharts classifies transitions according to the nature of risks they carry and, accordingly, extends the specification (labelling) of transitions with additional guards and enforcement conditions. Transitions belong to three categories: *safe* (hi-to-lo risk), *unsafe* (lo-to-hi risk) and *neutral* (between states of the same risk level). In terms of the function β, introduced informally in Section 3, this classification can be made as: a transition t is considered *safe* if $\beta(tg(t)) < \beta(sc(t))$, *unsafe* if $\beta(tg(t)) > \beta(sc(t))$, and *neutral* if $\beta(tg(t)) = \beta(sc(t))$. Thus, ε introduced in Section 4.1 triggers an unsafe transition, while μ triggers a safe transition.

Transition labelling in Safecharts has the general form $e\,[c]/a\,[l, u)\,\Psi[G]$, with e, c and a remaining the same as in Section 2 and certain components being mandatory depending on the risk classification of the transition concerned. $[l, u)$ is a right-open time interval from time l to time u. Ψ is a safety enforcement pattern specified using two alternative symbols: $\upharpoonleft$ and $\upharpoonright$, and $[G]$ is a safety clause. $t \upharpoonleft [G]$ is mandatory for unsafe transitions and means that the transition t is forbidden to execute as long as G holds. $t[l, u) \upharpoonright [G]$ is mandatory for safe transitions and means that the transition t is forced to execute within $[l, u)$ from whenever G begins to hold irrespective of the occurrence of its triggering event.

The *risk distance* of a transition $t \in T_S$ is the number of band boundaries between the source and target states of t in $\mathcal{G}(S)$. Denoting it by $\mathcal{D}(t)$, it can be defined as: $\mathcal{D}(t) = \beta(sc(t)) - \beta(tg(t))$, the positive and negative signs of $\mathcal{D}(t)$ thus signifying respectively an increasing, or decreasing, risk; see Figure 2(c). The risk nature of transitions plays an important role in determining their *safety enabling conditions*. For a neutral transition to be enabled, it must be *functionally enabled*, that is, its source state is active, its triggering event e has occurred and its guarding condition c, if any, is true. However, for an unsafe transition to be

enabled, it must be both functionally enabled and its safety clause G must be false. Likewise, for a safe transition to be enabled, it must be either functionally enabled or its safety clause G is true.

The enabling time of a transition t, denoted by $EnTime(t)$, is defined as the earliest time when t becomes safety enabled, as defined above. The time interval $[l, u)$ associated with safe transitions, introduced above, is a real-time constraint on t and imposes the condition that t does not execute until at least l time units have elapsed since it most recently became safety enabled, that is, since $EnTime(t)$, and must execute before u time units since $EnTime(t)$. If l and u have not been specified explicitly then t is assumed to be *spontaneous* with an open-ended $[0, 1)$ time interval. This implies that t executes as soon as it is enabled by G, in other words, as soon as $EnTime(t)$ is realised. In Safecharts, a transition t is *executed* if and only if it is safety enabled within its associated time interval and, either t is not in conflict with any other enabled transition or t has the highest priority among its conflicting transitions.

In Safecharts, two transitions t_1 and t_2 are said to be in conflict if $sc(t_1) = sc(t_2)$ and they become functionally enabled simultaneously. In Safecharts, nondeterministic choice between two, or more, conflicting transitions can be resolved by giving higher priority to the transition with the shortest risk distance. This approach is different from other approaches, for example in [13], where priorities are given according to the scope[1] of the conflicting transitions while in [4] priorities are given according to the hierarchy of their source states.

Let $conflict(t)$ be the set of all possible transitions which are in conflict with transition t. In the case of transitions with equal risk distances, prioritisation is based on the *cumulative* risk distances of future transitions of conflicting transitions. A transition t' is said to be a *future* transition of the transition t if the source state of t' is the target state of t. The set of future transitions of t can be defined as $future(t) = \{t' \mid sc(t') = tg(t)\}$. There is a greater likelihood of a future transition t' being executed if its source state becomes active as a result of the execution of a transition t among those in conflict, and its triggering event was generated by the execution of t. We refer to such transitions as *expected* future transitions and introduce the set: $expected(t) = \{t' \mid t' \in future(t) \wedge trg(t') \in gen(t)\}$.

Any nondeterminism between two, or more, conflicting transitions can now be resolved by giving highest priority to the competing transition with the smallest cumulative risk distance. The way cumulative risk distances of competing transitions are calculated is as follows: $\forall\, x \in (conflict(t) \cup \{t\})$
(1) if $expected(x) \neq \emptyset$ then select any transition y from the set $\{y \mid y \in expected(x) \wedge \forall\, y' \in expected(x) \Rightarrow \mathcal{D}(y) \leq \mathcal{D}(y')\}$
(2) if $expected(x) = \emptyset \wedge future(x) \neq \emptyset$ then select any transition y from the set $\{y \mid y \in future(x) \wedge \forall\, y' \in future(x) \Rightarrow \mathcal{D}(y) \geq \mathcal{D}(y')\}$
(3) In both above cases, resolve the non-determinism on the basis of $\mathcal{D}(x)+\mathcal{D}(y)$, otherwise on $\mathcal{D}(x)$ alone.

[1] The scope of a transition t is the lowest common OR ancestor state containing both $sc(t)$ and $tg(t)$.

Nevertheless, non-determinism may still continue to persist even after considering the future transitions as shown above, for example, if all, or some, transitions in $conflict(t)$ have equal accumulative risk distances. However, this kind of non-determinism is considered a *safe non-determinism* since all outcomes are identical in terms of the risks involved.

4.3 The Step Semantics of Safecharts

There exists many different semantics for Statecharts, centering mostly around the concept of *step*. The step semantics has been a much debated issue, primarily because of the anomalous and counter–intuitive behavioural patterns of Statecharts resulting from some of the interpretations. These debates concern the central issue as to whether the changes, such as the generated actions or updating values of data items that occur in a given step, should take effect in the current step or in the next step. The reader is referred to [15, 10, 11, 13] for more details about the different step semantics and the problems associated with their definitions.

In defining the step semantics of Safecharts, our aim here is to adopt the most appropriate standpoint in relation to the sole concern of Safecharts, namely the design of critical systems from whatever the perspective, whether it is from safety, security or any other system attribute. The step semantics of Safecharts retains certain characteristics of the conventional step semantics such as the synchronous hypothesis, while at the same time maintaining an intuitive relationship between external and internal events so that it corresponds to the operational reality of reactive systems. It is based on the treatment of external and internal events in an identical manner, but it also requires the introduction of the concept of *postponed transitions* and two separate notions of time, namely a *synchronous time* metric and a *real time* metric. The step semantics in Safecharts is based on the synchronous time model of STATEMATE [7]. The system evolves from one step to the next after considering a set of *input events* at consecutive intervals separated by a granularity of Δ time units, referred to as Δ-interval. The synchronous time model has the advantage of avoiding infinite loop of triggering transitions enabled by infinitely generated internal events, and preventing the occurrence of *racing conditions*.

The set of input events at the end of the current Δ-interval consists of the external events sent by the environment during the current interval as well as the internal events generated by the execution of the previous step. Input events last only for the duration of a single Δ-interval. Once the step has been taken, all input events are consumed and the set of input events becomes empty. In its initial state (initial configuration), the system waits for the environment to produce external events. At the end of the first Δ-interval, the input events consist of only the external events sent by the environment and are sensed and reacted to by executing the initial step. As a result of the initial step, the system moves to a new configuration, provided that the step is a 'status step' (in the sense discussed later), the generated internal events, if any, are added to the set of input events of the next step, and the clock is incremented by Δ-interval. The set of input events of the next step consists of the internal events, if any,

generated by the initial step together with the external events, if any, received by the environment during the following Δ-interval. In the example shown in Figure 5, the set of input events of *step1* consists of the internal event e_1 as well as the external events e_2 and e_3. At the end of the Δ-interval, *step1* is executed and all the input events are consumed. This process continues in each step.

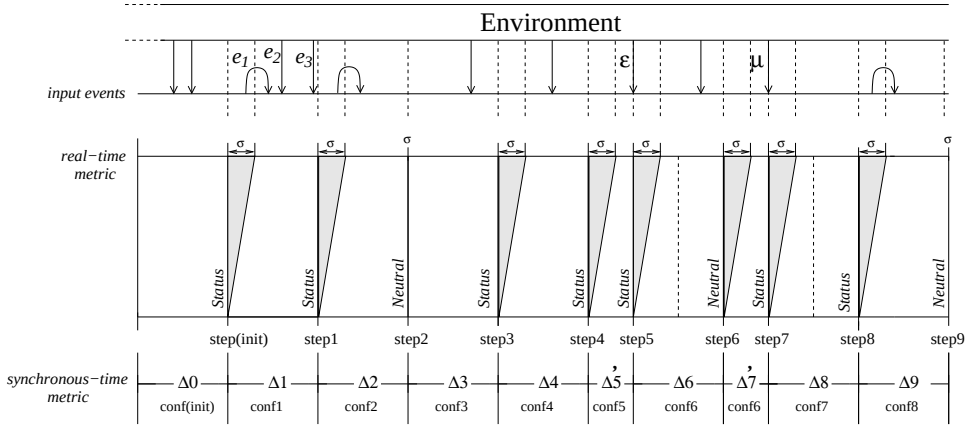

Fig. 5. The step semantics of Safecharts.

In the cases where there are no external events generated by the environment during the Δ-interval prior to the step, the set of input events comprises only the internal events generated in the previous step. In this case, the step is taken by consuming all input events and triggering relevant transitions. Consequently, internal events are possibly generated again for the next step, leading to a new configuration. In the case where there are neither external nor internal events from the previous step, that is, where the set of input events is empty, after Δ-interval the step is taken anyway without executing any transition and, consequently, with no change in the configuration of the system. For the system to move to a new configuration, the environment has to produce a new set of external events during the subsequent Δ-intervals. In this connection, our step semantics distinguishes two types of steps, namely *status* steps and *neutral* steps, the former causing a material change in the configuration of the system while the latter causing no change.

Analogous to several other definitions of step semantics, the step semantics of Safecharts eliminates many undesirable features, for example, negated events and instantaneous states. Safecharts also maintains a clear causality ordering and global consistency. Similar to the semantics of Statecharts introduced by [7] and adopted by many variants, the execution of a step in Safecharts takes zero time unit, and thus transitions triggered by input events are taken instantaneously once the step is taken. However, as stated in [8], the synchronous hypothesis does not reflect the intuitive operational reality of reactive systems, where transformations between the states of the system usually take some *real time*, during which the environment can send some external events. In order to

reconcile the mismatch between the synchronous hypothesis and the reality of transformational behaviour of real-time reactive systems, we propose two notions of time metrics: a *synchronous-time* metric and a *real-time* metric. In the synchronous-time metric, the duration of the step, denoted by σ, is always taken to be zero (in other words, σ is too fine to be detected), while in the real-time metric σ is either zero in the case of the step being neutral step, or a non-zero constant in the case of the step being a status step.

With reference to the real-time metric, the assumption underlying the adoption of the synchronous hypothesis is that, once a step is taken at the end of a Δ-interval, any external events sent by the environment during the σ time unit are postponed until the elapse of σ interval. Due to their importance in modelling the safety aspects of the system's behaviour, it is a feature in Safecharts that generic events, namely ε and μ, must be taken as soon as they occur. Thus, in this context, generic events are treated differently from other input events, and are considered as interrupt events. Once they occur and are added to the set of input events, the step does not wait until the end-time of the current Δ-interval, but rather executes immediately consuming all input events gathered so far. The Δ-interval during which generic events occur is called an *irregular* interval, and denoted by Δ'. The step that follows Δ' is called an *interrupt* step. For example, in Figure 5, $step_5$ and $step_7$ are two episodic steps executed as a result of the occurrence of events ε and μ respectively.

5 Case Study: Safecharts in the Security Domain

Alongside availability and reliability, safety and security are two closely related properties of dependable systems. The design of dependable systems is often required to satisfy several of these critical properties simultaneously. There is a growing interest in the degree to which techniques from one domain could complement, or conflict with, those from another. In this section, we investigate the applicability of Safecharts and its various safety-oriented techniques and mechanisms for dealing with security issues. More specifically, we examine the use of the concept of risk graph, and the various safety enforcement applied to transitions, in the field of security.

5.1 RBAC and its Modelling in Safecharts

In computer security, access control is the concept of managing authorisations, by which resources (objects) are accessed by individuals (subjects) with a specific set of operations. Role Base Access Control (RBAC) is a well-established approach in computer security for controlling access by users to various resources; see [14]. It is increasingly relevant to modern commercial, business and other domains. Our approach to modelling RBAC, however, is applicable to systems where security requirements are predominantly dependent on the state of the system. An exemplar of such systems is reactive systems which Statecharts and, hence, Safecharts are intended for. RBAC is based on the concept of *role* – a representation of job functions performed by individuals in an organisation [1]. Unlike in traditional access control mechanisms, such as those used in operating systems, RBAC assigns access rights to the roles rather than to the individuals directly. In

other words, the subjects are able to access objects only by virtue of their roles. A subject can be associated with more than one role and a role can be assigned to many subjects.

Permitted access rights of different roles to objects are maintained in an Access Control List (ACL) against which requests by subjects to perform various operations or tasks (e.g a *write* operation) on an object are checked. If a role authorising the access of the object concerned by the required operation is found, then the access right associated with this role is granted, otherwise, denied. The model of RBAC permits the temporary *delegation* of access rights by one party to another in order to perform one or more specific functions. Figure 6(a) depicts such a scenario in the context of an engineering organisation, where a manager A delegates some, or all, of his tasks (access rights) to a subordinate engineer Q, enabling Q to perform A's tasks on his behalf. This mechanism is vulnerable to potential security risks as ACL makes no distinction as to whether a subject requesting a certain mode of access is doing so in the capacity of his own role, for example, as originally assigned by the security officer, or in the capacity of a role acquired through a delegation.

As presented in [14], RBAC may be treated as a hierarchy of four models: RBAC_0 (flat model), RBAC_1 (hierarchical model), RBAC_2 (constrained model) and RBAC_3 (symmetric model). RBAC_0 is a model depicting simply various permissions allocated to various roles and, thereby, to different users. RBAC_1, on the other hand, depicts a seniority relation on roles, whereby senior roles automatically inherits the rights permitted to more junior roles to perform various tasks, reflecting the lines of authority or responsibility in a given organisation. Going further, RBAC_2 enforces separation of duties and RBAC_3 introduces the capability to review assignment of permission with changing circumstances.

In this work, we consider only the models, RBAC_0 and RBAC_1. RBAC involves generally three types of entities: users U, roles R and permissions P. Assignment of permissions (allocation of tasks) to roles is given by a function $\alpha \in R \to \mathbb{P}\, P$ so that, for any $r \in R$, $\alpha(r)$ gives the set of permissions assigned to the role r. Likewise, assignment of users to roles may be given by a function from U to $\mathbb{P}\, R$, though its detailed elaborartion is not required in this particular work. Since no restrictions are imposed on these functions, their representations are all that is required in RBAC_0. Turning to RBAC_1, in addition to its conventional features, our model considers here the relative risks associated with situations when users belonging to different roles performs different tasks. This is represented by risk ordering relations $\sqsubseteq$ and $\preccurlyeq$ on $R \times P$ with the same meaning as that given in Section 3.1. For example, the interpretation of $\preccurlyeq$ is such that for any $r_1, r_2 \in R$ and $p_1, p_2 \in P$, $(r_1, p_1) \preccurlyeq (r_2, p_2)$ is true if and only if the security risk level associated with a user in role r_1 performing the task p_1 is known to be strictly lower than that of with a user in role r_2 performing a task p_2, unless r_1 and r_2, and p_1 and p_2, each denote the same entity. Let $\leqslant$ denote the hierarchical ordering on R such that, for any $r_1, r_2 \in R$, $r_1 \leqslant r_2$ is true if and only if the role r_1 is of a lower, or an identical, hierarchy compared to the role r_2. In our model, $r_1 \leqslant r_2$ if and only if

$$\alpha(r_1) \subseteq \alpha(r_2) \wedge (\forall\, p \in P \bullet p \notin \alpha(r_1) \wedge p \in \alpha(r_2) \Rightarrow (r_2, p) \preccurlyeq (r_1, p)) \qquad (13)$$

In other words, each role of any given higher hierarchy consists of some specific tasks not permitted by the roles of the lower hierarchies on security grounds, for example, based on criteria such as trustworthiness, required competence level and so forth. The above thus expresses in our model a principle of *permission assignment* to roles. This is a capability not found in the formal representation of conventional RBAC_1 [14]. In fact, classification of role hierarchies has to be based on some sort of risk assessment and, in this respect, the concept of risk graph in Safecharts provides a formal basis for achieving this. In the remainder of this section, we illustrate the use of Safecharts in modelling temporary delegation of a higher rank role to a user of a lower rank role in RBAC and the use of the principle (13) in establishing assignment of permissions to roles and, hence, the determination of enforcement conditions appearing in certain transition labels.

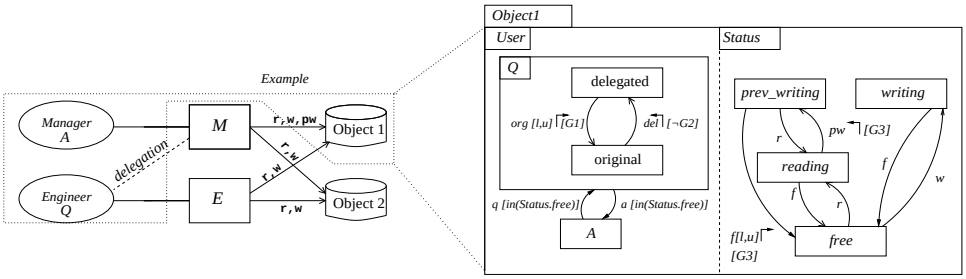

Fig. 6. An example of delegation in RBAC and the safety layer.

Figure 6(a) shows the example being considered – a scenario involving just two users, a manager A in role M and an engineer Q in role E, accessing a particular object O, for example, a file or a database. The following operations are permitted: *read* (read only), *write* (both read and write), *priv_writing* (privilege writing from a read–only mode). The two roles are such that $E \preccurlyeq M$, though the actual role–permission relation shown in the figure is to be established later. Figure 6(b) shows a Safecharts model for part of it, depicting how the two users access *Object*1, modelled as an AND state, via their respective roles. Substates of *Object*1 are two OR states: *Status* showing possible states the object can be in, and *User* signifying that the object can be accessed by the two users in an exclusive mode. The order in which the states are placed vertically in the diagram of any OR state corresponds to an implicit risk ordering. For example, the object being in the state *free* is considered safer (more secure) than being in the state *reading*, while being in the state *reading* safer than being in the states *writing* or *priv_writing*. In contrast, states *writing* and *priv_writing* are assumed to be risk non-comparable. In other words, a risk ordering of the form (*free* $\preccurlyeq$ *reading*), (*reading* $\preccurlyeq$ *writing*) and (*reading* $\preccurlyeq$ *pre_writing*) is assumed in *Status*. Similarly,

for the state Q, a risk ordering of the form $(original \preccurlyeq delegated)$ is assumed, indicating that the user Q accessing the object in his original role is safer than accessing the same object in a delegated capacity. Accordingly, the transition $original \rightsquigarrow delegated$, signifying the delegation of a role, is an unsafe (unsecure) transition, whereas the transition $delegated \rightsquigarrow original$, signifying revocation of the role delegation, is a safe (secure) transition.

Though the risk ordering described above has some significance in an isolated context, it is not adequate for describing the risks involved in access control. This is because the interdependencies of risks, depending on the roles of the users involved and the operations being performed by them, need to be considered. In other words, we need to consider risks posed by different combinations of states. Technically speaking, this amounts to flattening of the AND state $Object1$ into an equivalent OR state and developing a risk graph for the flattened state. As was mentioned in Section 3.2, the required risk graph can be constructed either directly or indirectly; in this example, we follow the former. The set of sub-states of the resulting equivalent OR state and, hence, the nodes of the resulting risk graph, consists of eight (pairs of) states: $(Q, writing)$, $(Q, prev\_writing)$, $(Q, reading)$, $(Q, free)$, $(A, writing)$, $(A, prev\_writing)$, $(A, reading)$, $(A, free)$. Furthermore, since Q is itself an OR state, each node involving Q in the risk graph needs to be refined and replaced by its risk graph, that is $\mathcal{G}(Q)$. As a result, the node $(Q, writing)$, for example, must be replaced by a risk graph consisting of two nodes, possibly, with a risk ordering of the form $(Q.delegated, writing) \preccurlyeq (Q.original, writing)$. Hence, the resulting risk graph will consist of a total twelve nodes.

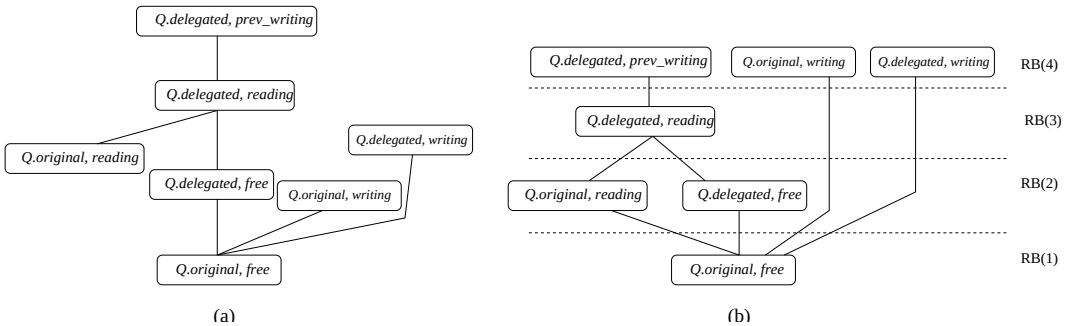

Fig. 7. The risk ordering relation and the risk graph of the user Q.

The direct approach adopted here to construct the risk graph for the flattened state provides an opportunity to review the risks involved, ideally, based on a proper security analysis of the problem concerned. For illustrative purposes, we have assumed that $(A, priv\_writing) \prec (Q.original, priv\_writing)$; no other distinctions are being made otherwise between A and Q when Q is acting in his original role. For reasons of space, we have also chosen to concentrate here only on the behaviour of Q and, thus limiting ourselves to the risk graph shown in Figure 7(a) giving risk ordering related to Q only. Note

that the states $(Q.original, writing)$ and $(Q.delegated, writing)$ are treated as non-comparable by $\sqsubseteq$ with many other states (nodes). As a precaution against this being possibly due to an inadequacy of the risk assessment process, the banded risk graph in Figure 7(b) has placed these states conservatively in the highest risk band. This is to be taken as a flag, alerting the designer to reconsider the risk levels of these states if the circumstances do not warrant such an interpretation. Due to the positions of their source and target states in the risk graph, transitions such as $(Q.original, free) \rightsquigarrow (Q.original, writing)$, $(Q.original, free) \rightsquigarrow (Q.delegated, writing)$ and $(Q.original, free) \rightsquigarrow (Q.delegated, prev\_writing)$ have equal risk distances. An implication of this is that in the event of a conflict, selecting either of these transitions is considered as a safe non-determinism. If this is unacceptable on security grounds, then the designer needs to verify the security policy and/or the relative risk levels of their target states, namely $(Q.original, writing)$, $(Q.delegated, writing)$ and $(Q.delegated, prev\_writing)$.

Turning to the permission assignment, the two roles can now be distinguished in accordance with the principle (13). It can be seen, for example, that $\alpha(E) = \{read, write\}$ and $\alpha(M) = \{read, write, priv\_writing\}$ satisfies (13). As a result, a user belonging to the role E will no longer have access to the operation *priv_writing* in his *original* capacity. However, this can be allowed in a delegated capacity provided that he satisfies a suitably specified condition G_2 expressing, perhaps, that additional measures have been taken to ensure his temporary security credentials, for example, that he has been given additional training. Hence, the prohibition condition $(\neg\ G_2)$ appearing in the label of the transition $original \rightsquigarrow delegated$ and the enforcement condition G_1 appearing in that of $delegated \rightsquigarrow original$, G_1 being identical to G_2 or being a timeout. Likewise, a pair of prohibition and enforcement conditions involving a predicate G_3, G_3 being defined as $in(Q.original)$, have been added to the labels of the transitions $reading \rightsquigarrow prev\_writing$ and $priv\_writing \rightsquigarrow free$ respectively. Thus, role classification arrived above allows us to deal with temporary delegation of tasks of senior roles to individuals of more junior roles. The above is a systematic approach for avoiding problems such as the one mentioned earlier in relation to the conventional implementation of ACL.

6 Conclusion

Correct interpretation of Safecharts requires a sound understanding of several important aspects of its semantics. The objective of this paper has been to present a mathematical framework for Safecharts, with the primary aim of giving greater clarity and accuracy to key concepts used in Safecharts. These include separation of function and safety, risk ordering, the risk graph, failures, risk nature of transitions and resolution of non-determinism. In extending the notion of risk ordering to include composite OR and AND states, the paper also extends previous work by showing how to deal with interdependencies of risk at lower levels of abstraction; this involves flattening AND states and node replacement in risk graphs. In this paper, the step semantics of Safecharts has been defined and the mismatch between the synchronous hypothesis and the reality of trans-

formational behaviour of real-time reactive systems has been reconciled. Rules on resolution of non-determinism between any conflicting transitions have also been refined to include triggering events of transitions. Although Safecharts was originally developed explicitly for safety-critical systems design, its various features and mechanisms used to ensure safety are found to be equally valid in the security domain. This has been demonstrated in this paper using, for illustrative purposes, a simple but realistic example of delegation of roles as understood in Role Based Access Control (RBAC) in security. On–going work investigates *situational events* – a special kind of events which calls for alteration to the risk ordering relation dynamically to account for unfolding scenarios brought about by a chain of failure events.

References

1. Barka E. and Sandhu R. *Framework for Role-Based Delegation Models.* Proceedings of the 16th IEEE Annual Computer Security Applications Conference, pp: 168–175, New Orleans, Louisiana, USA, December, 2000.
2. Dammag H. and Nissanke N. *Safecharts for specifying and designing safety critical systems.* Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems, pp: 78–87, Lausanne, Switzerland, 1999.
3. Dammag H. and Nissanke N. *A Mathematical Definition for Safecharts.* Technical Report, South Bank University, SBU-CISM-03-02 March, 2003.
4. Day N. *A Model Checker for Statecharts (Linking CASE tools with formal Methods).* Technical Report 93-35, University of British Columbia , Vancouver, Canada, 1993.
5. Glinz, M. *An Integrated Formal Model of Scenarios Based on Statecharts.* LNCS, Vol. 989, pp: 254–271, Springer, 1995.
6. Harel D. *Statecharts: A Visual Formalism For Complex Systems.* Science of Computer Programming, Vol. 8, No. 3, pp: 231–274, North-Holland 1987.
7. Harel D. and Naamad A. *The STATEMATE Semantics of Statecharts.* In ACM Transactions on Software Engineering and Methodology, Vol. 5, Issue 4, pp: 293–333, October 1996.
8. Huizing C. and de Roever W. *Introduction to Design Choices in the Semantics of Statecharts.* Information Processing Letters, Vol. 37, Issue 4, pp: 205–213, 1991.
9. Hong H. S., Kim J. H, Cha S. D and Kwon Y. R. *Static Semantics and Priority Schemes for Statecharts.* Proceedings of the 19th International Computer Software and Applications Conference COMPSAC'95, pp: 114–120, Texas, USA, 1995.
10. Lüttgen G. and Mendler M. *The Intuitionism Behind Statecharts Steps.* In ACM Transactions on Computational Logic, Vol. 3, Issue 1, pp: 1–41, 2002.
11. Maggiolo-Scheltini A., Peron A. and Tini S. *A Comparison of Statecharts Step Semantics.* Theoretical Computer Science Journal, Vol. 290, Issue 1, pp: 465–498, January, 2003.
12. Nissanke N. and Dammag .H *Design for safety in Safecharts with risk ordering of states.* Safety Science, Vol. 40, Issue 9, pp: 753–763, December 2002.
13. Pnueli A. and Shalev A. *What is in a Step: On the Semantics of Statecharts.* In Ito T. and Meyer A.R., editors, Theoretical Aspects of Computer Software, LNCS 526, pp: 244–265. Springer, 1991.
14. Sandhu R., Ferraiolo D. and Kuhn R. *The NIST Model for Role-Based Access Control: Towards A Uinfied Standard.* In Proceedings of 5th ACM Workshop on Role-Based Access Control, pp: 47–64, Berlin, Germany, July, 2000.
15. Von der Beek M. *A Comparison of Statecharts variants.* LNCS 863, pp: 128–148, Springer, Berlin, 1996.