Using Safecharts For Modelling Security Issues

Hamdan Dammag Nimal Nissanke Etienne Khayat
London South Bank University
103 Borough Road
London SE1 0AA, UK
{dammagh,nissanke,e.khayat}@lsbu.ac.uk

Abstract— **This paper introduces a novel security–oriented interpretation of Safecharts – a visual formalism originally developed for safety–critical systems design. Safecharts itself is based on Harel's Statecharts and draws on its unique features for representing risks against safety due to circumstances equipment failures. One of these features is the the risk ordering relation enabling the provision of additional safeguards for ensuring safety. Built into the risk ordering relation are conservative assumptions with respect to any inadequacies in the risk assessment process. Despite its sole use so far for dealing with safety, there is no reason why the risk ordering relation cannot be used to express requirements on any other system property, including security. The aim of this paper is to show how this can be achieved and to show that the features of Safecharts have a natural interpretation in a security context with similar virtues as in safety. A further contribution outlined here is the newly developed step semantics of Safecharts central to modelling of safety and security requirements.**

I. INTRODUCTION

Computer systems are increasingly used in areas where their failures could have serious consequences in terms of potential losses. The nature of these losses varies, ranging from threats to material assets (property, finance and information) to human life itself. In this respect, there are many properties such systems should possess, having a profound influence on the techniques used in their development, as well as on the manner of their operation and maintenance. Among them are safety and security which, alongside availability and reliability, form two defining characteristics of dependable systems. Conceptually, safety and security have many commonalities between them, for example, both properties deal with *threats* or *risks*, one to life and property and the other to privacy or organisational or national security [8]. However, safety and security differ from each other in the nature of failures each is concerned with, accidental (unintentional) failures in the case of the former and malicious attacks (intentional failures) in the case of the latter.

It is usually the case that in practice safety and security issues are dealt with largely in isolation. Perhaps as a result, research into computer safety and computer security too have followed historically separate paths. However, there is a growing realisation about the benefits to be drawn from a greater understanding of the two domains, in particular, their complementary features and the areas of divergence. This could be turned into mutual advantage by borrowing ideas effective in each other and providing alternative but complementary perspectives.

In this context, this paper investigates the use of Safecharts [3], a safety–oriented variants of Statecharts developed especially for the specification and modelling of safety–critical systems, in the field of security. Unique features of Safecharts include the maintenance of a clear distinction between functional and safety requirements, an explicit representation of failures, mechanisms for handling them, a safety–oriented classification of transitions and resolution of any conflict between them in favour of safer transitions. Fundamental to the above is an explicit ordering of system states according to their risk levels. This is achieved through a risk ordering (mathematical) relation and a concept called risk graph, the latter being an approach to ensuring conservative assumptions about any state not covered adequately by the risk assessment process. Motivation for this paper is the relaisation that the above risk ordering relation has a more general interpretation and is not restricted to safety. For example, the risks posed may concern threats against security, in which case various features of Safecharts could be given a security–oriented interpretation. This kind of interpretation could lead to extended applications of Safecharts in totally different contexts, security being one of them.

With the above as an objective, this paper illustrates the use of Safecharts in demonstrating the risk involved in security policies and requirements, more specifically, in the field of access control of data security. The Role–Based Access Control (RBAC) mechanism is adopted for demonstrative purposes. RBAC is a well-known access control mechanism that bases access control decisions with respect to functions, which users are allowed to perform within an organization. An immediate benefit is the more intuitive visual appreciation of the underlying security model. At a technical level, benefits include a systematic way to counter breaches of security, a way to specify security requirements on individual transitions by either prohibiting or forcing their execution, secure initialisation of states, security–oriented resolution of any unforeseen nondeterministic execution of transitions and prohibition of introducing transitions between states posing unknown security threat levels on precautionary grounds. These are all designed to enhance two fundamental properties of computer security, namely confidentiality and integrity.

With this general interpretation of risk, while maintaining the same basic analytical and modelling framework, this paper lays, in essence, the foundation for a single integrated framework for dealing with any combination of system properties relevant to dependable systems. At this stage, this capability is limited to safety and security only. This paper also addresses several novel issues concerning the semantics of Safecharts, in particular, the construction of risk graphs for AND states and the definition of its step semantics, especially in relation to its other unique features.

II. Safecharts

A. Statecharts in Brief

Statecharts is a visual specification formalism introduced by David Harel [5] 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 broadcast 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. BASIC states are similar to the states in state-transition diagrams. 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, state S is an AND

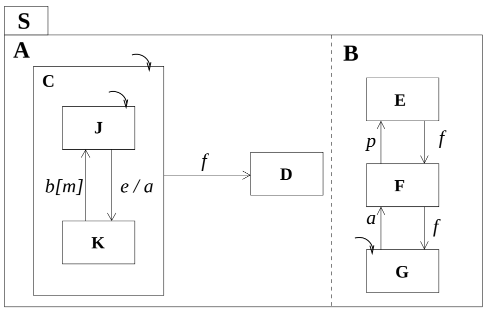

Fig. 1. An example of Statecharts.

state with two (orthogonal) substates A and B, the latter two states 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 Figurer 1 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$\rightsquigarrow$F, moving the system to state F. As a result, a new configuration of state S is realised, namely $\{$K, F$\}$.

B. Modelling Safety and Security in Safecharts

Although Safecharts was introduced for addressing safety concerns, security issues can be also addressed in the framework of Safecharts in the same manner. Thus, Safecharts features in this work could refer to either of these system attributes: safety or security. One of the unique features of Safecharts is the maintenance of two separate layers of representation. In the safety domain, they have been referred to as *functional layer* and *safety layer*. The same term is often used in relation to security, despite *security layer* being the right term to be used in this context. Likewise, unless otherwise stated, any statement made on safety applies equally to security, and vice versa. The aim of the functional layer is to capture system's transformational behaviour purely from a functional point of view, by using Statecharts in the conventional sense. Conversely, the aim of the safety layer is to capture the risk involved in such behaviour. It contains a *risk graph* of the system's states and a safety annotation associated with any transition between these states. The separation drawn in this manner between function and safety properties of the system helps in (i) focusing on safety matters without being distracted by functional issues, (ii) evaluating the implications of function on safety and (iii) ensuring safety provisions for each and every action involving any risk. Another unique feature of Safecharts, especially in the safety domain, is its explicit representation of failures by means of two generic events, namely ε and μ and some special states denoting the different failure modes of the system. The ε event signifies a nondeterministic failure and μ event signifies a recovery action, e.g. a repair of a faulty component. For more details about the above features, the reader is referred to [3], [10].

C. The Risk Graph

Fundamental to Safecharts semantics is the explicit ordering of system states according to their risk levels. This is achieved through a risk ordering relation, denoted by $\sqsubseteq$, and a concept called risk graph. The definition of $\sqsubseteq$ is that for any two states s_1 and s_2, $s_1 \sqsubseteq s_2$ 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. This assumes that risk levels are comparable, either quantitatively or qualitatively. The relation $\sqsubseteq$ can be decomposed into two relations: a partial order relation $\preccurlyeq$ (the risk level of s_1 being strictly lower than, or being the same as, that of s_2) and an equivalence relation $\approx$ (risk levels of s_1 and s_2 are known, or are assumed, to be identical).

The concept of risk graph incorporates conservative assumptions about states not covered adequately by the risk ordering relation $\sqsubseteq$, possibly due to gaps in the risk assessment process. This can be a result of oversight, omissions or lack of knowledge about the relative risk levels of these states. An incomplete risk assessment process, for example, could result in a set of states not receiving adequate attention and not being comparable with other states. This might result in some states being non–comparable by the $\sqsubseteq$ relation with a large number of other mutually comparable states. Figure 2(a) gives an example, where state H happens to be non–comparable with states D, G, I and J, and state D is non–comparable with states I, G and H.

The risk graph enhances the risk ordering relation $\sqsubseteq$ by applying the concept of *risk bands* such that each state in the risk graph belongs to a unique risk band and every pair of distinct states belonging to the same risk band is there either because the states concerned are comparable by the $\approx$ relation or non–comparable by $\sqsubseteq$, the latter kind of states are known as *risk-noncomparable* states. Given the risk ordering relation $\sqsubseteq$, and assuming that risk bands are indexed numerically from 1 to some n, risk bands of states may defined according to the following set of rules:
(1) States in the highest risk band n consists of exactly (a) maximal elements (states) in the partial order relation $\preccurlyeq$ but excluding those elements, if any, which are comparable by $\approx$ with any of the rest of elements in $\preccurlyeq$, and (b) elements which are comparable by $\approx$ with those elements defined in (a) above.
(2) Any state s with just a single immediate (distinct) successor state, which is in risk band i according to $\preccurlyeq$, is in risk band (i-1). However, if a state s has more than one immediate successor state, then it has a risk band one less than the lowest of the risk bands of its immediate successor states.
(3) For any states s_1 and s_2, if $s_1 \approx s_2$ then both states s_1 and s_2 are in the same risk band.

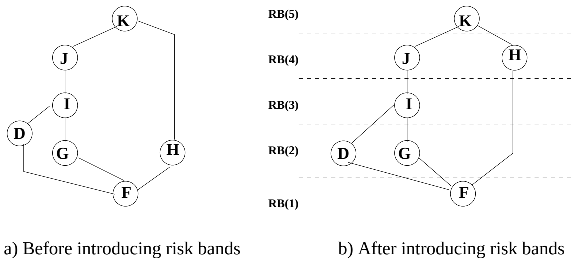
K
J
I
D
G
H
F
a) Before introducing risk bands
RB(5)
RB(4)
RB(3)
RB(2)
RB(1)
K
J
H
I
D
G
F
b) After introducing risk bands

Fig. 2. Risk bands in the risk graph.

Figure 2(b) illustrates the above rules using the banded version of the risk graph given in Figure 2(a). Note that as a consequence of the position of state H in the risk graph, state H can be now compared to any state that belongs to a different band. In other words, state H can be considered as a higher risk state than states F, D, G and I. However, state H and state J remain risk-noncomparable.

D. Risk Graph of AND state

Unlike OR states, the construction of risk graphs of AND states is complicated because of the presence of orthogonal states. In an AND state, risk ordering in an individual orthogonal child OR state, without paying due regard to risk ordering in adjoining child OR states, no longer makes sense. Thus, a risk graph of an AND state S can be constructed using two alternative approaches:

1) Directly, that is, explicitly through a risk assessment based on the states (tuples) of an equivalent *flattened* OR state S'.
2) Indirectly, that is, using a subsidiary risk ordering relation $\sqsubseteq_\beta$ defined in terms of the risk band indices of individual orthogonal risk graphs. This is a process which does not require the security engineer to flatten the AND state in defining the risk ordering relation for the AND state concerned as a whole.

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 [4]. 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 derived transitions.

In the case of the direct approach, the risk ordering relation $\sqsubseteq$ can be defined over the set C'. The risk graph is then constructed in the same manner as described in Section II-C. In the case of the indirect approach, the subsidiary risk ordering relation $\sqsubseteq_\beta$ is defined first and then the risk ordering relation $\sqsubseteq$ is obtained by picking the respective states residing in the corresponding risk bands specified in $\sqsubseteq_\beta$. In this paper, we adopt the direct approach.

E. Transitions and Nondeterminism

Based on the risk graph, Safecharts classifies transitions according to the nature of risks they carry and, accordingly, extends the labelling of transitions with additional guards and enforcement conditions. Thus, transitions belong to three categories: *safe* or *secure* (from a higher risk state to lower risk state), *unsafe* or *unsecure* (from a lower risk state to higher risk state) and *neutral* (between states of the same risk level). Being an exhaustive classification, Safecharts disallows transitions between risk-noncomparable states. The reasoning behind this principle is prudence and it is intended to prompt the designer to resolve, as a matter of discipline, the risk levels of any non–comparable states belonging to the same risk band in the risk graph, if a transition is desired between them.

Transition labelling in Safecharts has the general form $e[c]/a\ [l,u)\Psi[\mathcal{G}]$, with e, c and a remaining the same as understood in Statecharts. The $e[c]/a$ part of the label is associated with the transitions represented in the functional layer. The $[l,u)$ is a right-open time interval from a lower bound l to an upper bound u. The Ψ is a safety or security enforcement pattern imposed on the execution of the transition. $\mathcal{G}$ is a safety or security clause; a predicate that specifies under which conditions the transition must, or must not, execute. The Ψ stands for one of the alternative symbols $\Lsh$ and $\Rsh$, and signifies one of the following enforcements: (a) a *prohibition* enforcement, denoted by $\Lsh$, such that given an unsafe transition t, the label $t \Lsh [\mathcal{G}]$ signifies that, despite

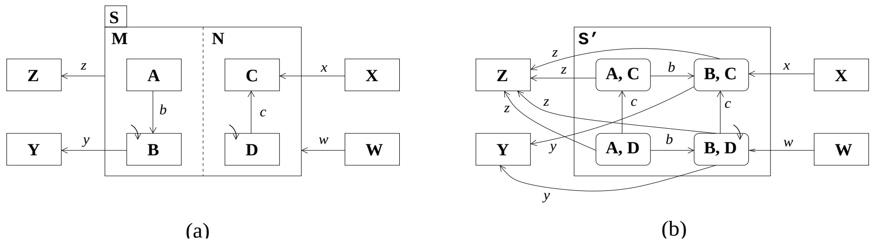

Fig. 3. An AND state and its equivalent flattened OR state.

being enabled by its triggering event, the transition t is forbidden to execute as long as $\mathcal{G}$ holds; (b) a *mandatory* enforcement, denoted by $\Rsh$, such that given a safe transition t, the label $t[l,u) \Rsh [\mathcal{G}]$ signifies that, irrespective of the occurrence of its triggering event, the transition t is forced to take place within the time interval $[l,u)$ whenever $\mathcal{G}$ holds. If the interval $[l,u)$ was not explicitly specified then t is assumed to be *spontaneous* and is forced to take place as soon as G holds.

Transitions in Safecharts are associated with *risk distances*. The risk distance of a given transition is calculated by subtracting the risk band index of the source state from that of the target state. It follows from the above that safe transitions have negative risk distances, unsafe transitions positive risk distances and neutral transitions zero risk distances. In Figure 2(b), an unsafe transition F$\rightsquigarrow$H will have a risk distance of $+3$, while a safe transition H$\rightsquigarrow$G will have a risk distance of -2. Risk distances are used for prioritising conflicting transitions. Two transitions are said to be in conflict if they share the same source state and their triggering events occurred at the same time. Safecharts resolves the nondeterministic choice between cinflicting transitions by using their risk distance such that the smaller the risk distance of a transition the higher is its priority. If two transitions have the same risk distance, Safecharts evaluates their cumulative risk distances by considering their *future* transitions, for more details see [3], [10]. Nondeterminism may still continue to persist even with future transitions. This kind of nondeterminism is considered a safe (secure) nondeterminism since all outcomes are identical in terms of the risks involved.

III. 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 [2], [9], [11] for more details about the different step semantics and the problems associated with their definitions.

It is important in this respect to define the step semantics of Safecharts in a clear and simple manner. The 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 behaviour of the step is associated with a clock that maps the time values belonging to the above two time metrics. The step semantics in Safecharts is based on the synchronous time model of STATEMATE [6]. 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 (a) avoiding infinite loop of triggering transitions enabled by infinitely generated internal events, and (b) 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 (configuration), the system waits for Δ-interval for the environment to produce external events. At the end of the Δ-interval, the input events, which at this stage consist of only the external events sent by the environment, 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 preceding Δ-interval. For example in Figure 4, the set of input events of *step1* consists of the internal event e_1, generated by *step$_{init}$*,

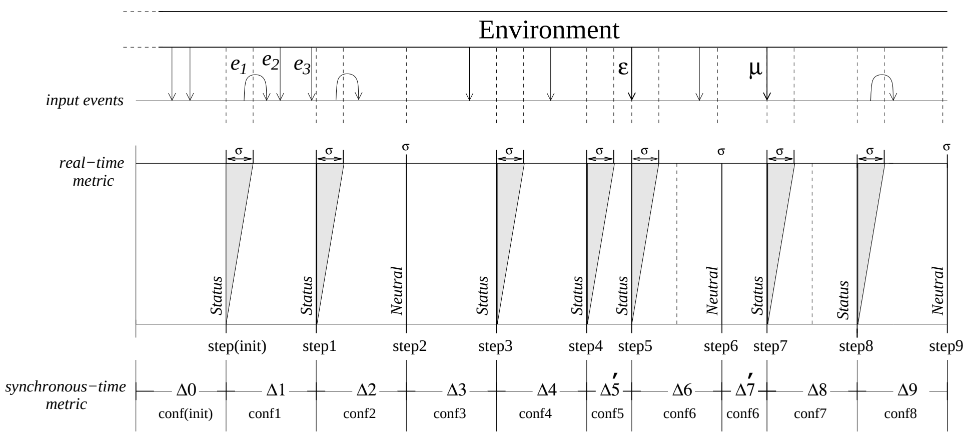

Fig. 4. A general view of the step semantics with generic events.

as well as the external events e_2 and e_3. At the end of the Δ_1-interval, *step1* is executed and all the input events are consumed. This process continues in each step.

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, new internal events might be possibly generated 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 new 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. This concerns the use of negated events and instantaneous states. Safecharts also maintains a clear causality ordering and global consistency. Similar to the semantics of Statecharts introduced in [6] and adopted by many other 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 [7], 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 reactive systems, we propose two notions of time metrics: a *synchronous-time* metric and a *real-time* metric. The former is a concrete time which is represented by a system clock that is constructed by a sequence of Δ-intervals, while the latter is a ‘true time’ which is independent from the representation of the system clock. Consequently, the evolution of our step can be seen from two different perspectives: the synchronous-time metric and the real-time metric reflecting the temporal reality of the transformations between the states of the systems. Accordingly, 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 (e.g. equipment failures and breaches of security), 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 being 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. The behaviour of interrupt steps is similar to that of other steps except that their execution time is not scheduled but rather occurs as a result of an interrupt situation. For example, in Figure 4, $step_5$ and $step_7$ are two interrupt steps executed as a result of the occurrence of events ε and μ respectively.

IV. Safety and Security

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 [13]. There is a growing interest in the degree to which techniques from one domain could complement, or conflict with, those from another. Possible interrelationships between safety and security have been the objective of much research, most of which has been in the area of incorporating security techniques in the safety domain. This includes the early work [12] on possible uses of security kernels in relation to safety. Another example is the work [15], where a non–interference concept, used in describing security properties was used in describing safety. However, reliability oriented safety mechanisms such as fault-tolerance have also been used in the field of security. For example, in [16], the possibility of extending fault-tolerance techniques to tolerate intentional attacks rather than accidental faults has been investigated. In this paper, 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 data security. Note that our discussion in this section applies to security only.

*A. Delegation in Role–Based Access Control (*RBAC*)*

In computer security, access control is the concept of managing authorisations, by which resources (objects) are accessed by users (subjects) under a specified set of operations. Subjects can access objects under the rules stated by the access control mechanism that describes the security policy of their organization. RBAC is a well-known mechanism that provides a high level view of access control. RBAC is based on the idea of *role* – a representation of job functions a subject performs in an organisation [14]. Unlike in traditional access control mechanisms, such as those used in operating systems, RBAC assigns access rights to the roles rather than to individual subjects directly. In other words, the subjects are able to access the 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.

Authorised access rights of different roles to objects is maintained in a form similar to an Access Control List, 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, the access right associated with this role is granted to the subject requesting it, otherwise, the access right is denied. The model of RBAC permits the temporary *delegation* of access rights by one subject to another in order to perform one or more specific functions. Figure 5(a) depicts such a scenario in the context of an engineering organisation, where a manager *P* delegates some, or all, of his authorised tasks (access rights) to a subordinate engineer *Q*, enabling *Q* to perform *P*'s tasks on his behalf. The manager *P*, known as the delegating subject, can claim back his role by a process known as *revocation*, the case in which the engineer *Q*, known as the delegated subject, loses his association with the manager role, and thus, all access rights granted by that role.

The process of delegation is vulnerable to potential security risks because there is 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. For example, in Figure 5(a), the manager's role, denoted by *M_Role*, is authorised to access *Object*1 with the *write* and *read* operations and to access *Object*2 with only the *read* operation. The engineer's role, denoted by *E_Role* is authorised to access *Object*2 with the *write* and *read* operations but cannot access *Object*1. In this case, from the object's point of view, there is no demarcation between the cases when *Object*1 is accessed by a manager using his original role, that is *M_Role*, and when it is accessed by an engineer exercising the same role in a delegated capacity. The risks of such delegation lie in potential 'undesirable behavioural patterns' of the delegated subject. Although such a behaviour could be unintentional in most cases, it still carries a degree of risk, possibly, due to the lack of competence of the delegated subject, his unfamiliarity with the delegated task and/or sensitivity of the delegated task itself, giving rise to a greater likelihood of errors. Such risks may not be easily quantifiable or assessed.

B. A Case Study

In order to illustrate the use of Safecharts in modelling a delegation scenario in RBAC , let us consider, for reasons of space, a small part of the diagram shown in Figure 5(a), namely the relationship between the *Object*1 and the *M_Role*. The delegation used in our example is assumed to be *temporary*, i.e. the subject is delegated a role for a certain period of time, after which the role is revoked, and *total*, i.e. the delegation includes all access rights associated with the role being delegated. We also assume a *grant dependent* revocation, i.e. the delegated role can be only revoked by the delegating subject [1]. Moreover, after delegating it to another subject, the delegating subject cannot use his original role unless the delegation is revoked.

The safety (security) layer of Safecharts for the example above is shown in Figure 5(b). The *Object*1 is represented as an AND state, consisting of two orthogonal states, namely *Status* and *M_Role*. The former represents the status of the object in terms of the operations applicable to it, while the latter represents the different capacities of *M_Role* in which the object can be accessed. The state *Status* consists of three substates, namely *written*, *read* and *idle*, each state denoting that the object is being written in to, is being read and is in idle state respectively. The state *M_Role* consists of two substates, namely *delegated* and *original*, denoting whether the object is being accessed by a delegated role or by an original role respectively. According to this model, the object can be in any of its status states, for example in *idle*, while its accessiblity is being determined by any of the two states: *delegated* or *original*.

Note that the order in which states are placed in the Safecharts diagram vertically corresponds to an implicit risk ordering. For example, the object being in state *idle* is considered more secure than being in state *read*, and that being in state *read* is more secure than being in state *written*. In other

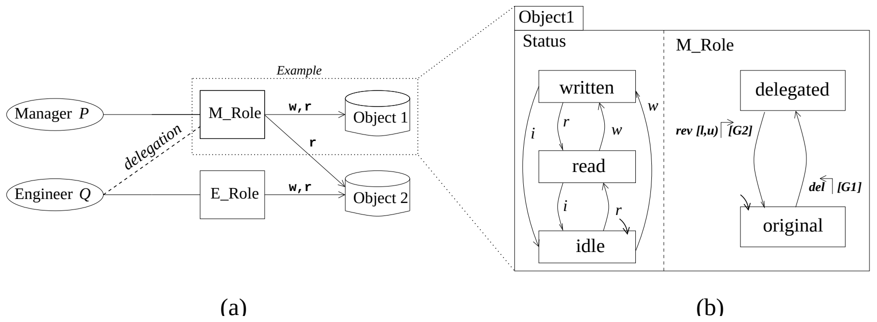

Fig. 5. An example of delegation in RBAC and the security layer.

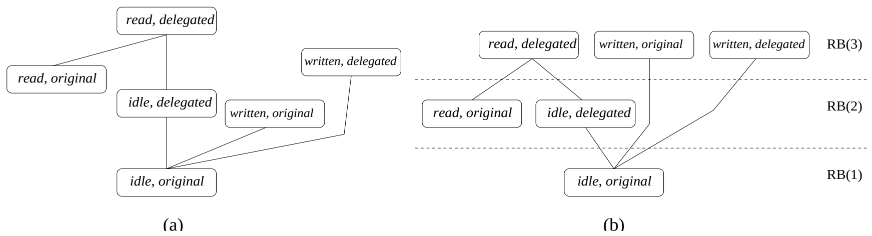

Fig. 6. The risk ordering relation and the risk graph of Object1.

words, a risk ordering of the form (*idle* $\preccurlyeq$ *read* $\preccurlyeq$ *written*) is assumed. Similarly, the ordering of states *delegated* and *original* inside state *M_Role* assumes a risk ordering of the form (*original* $\preccurlyeq$ *delegated*), indicating that the object being accessed by an original role is considered more secure than being accessed by a delegated role. In Figure 5(b), states *idle* and *original* are the *secure* default states of states *Status* and *M_Role* respectively. Thus, at initialization, the object is to be set to its secure default state, that is, to the configuration (*idle, original*). The transition *original*$\rightsquigarrow$*delegated* in Figure 5(b) is an unsafe (unsecure) transition with a prohibition enforcement $\Lsh$ $[\mathcal{G}_1]$. This means that under certain security concerns, denoted by the security clause $\mathcal{G}_1$, the transition is prohibited from taking place despite the occurrence of its triggering event *del*. For example, $\mathcal{G}_1$ can be a statement in the security policy denying such delegation. However, the transition *delegated*$\rightsquigarrow$*original* is a safe (secure) transition with a mandatory enforcement $[l, u)$ $\Rsh$ $[\mathcal{G}_2]$. This means that under certain security concerns, denoted by the security clause $\mathcal{G}_2$, the transition, representing a revocation of the delegated role, is forced to take place within a specified time interval $[l, u)$ despite the absence of its triggering event *rev*. For example, $\mathcal{G}_2$ can be the occurrence of the 'undesirable behaviour' mentioned in Section IV-A.

In order to construct its combined risk graph, i.e to show in detail the risks posed by different combinations of states, we use the direct approach, introduced in Section II-D, which involves flattening the AND state *Object1* to its equivalent OR state. The set of substates of the equivalent OR state, resulting from such a direct specification of relative risk levels, consists of six states (pairs), namely $\{$(*idle, original*), (*idle, delegated*), (*read, original*), (*read, delegated*), (*written, original*), (*written, delegated*)$\}$. Equivalently, as an assessment of relative risk levels of such states can be assumed: (*idle, original*) $\preccurlyeq$ (*idle, delegated*), (*idle, original*) $\preccurlyeq$ (*written, original*), (*idle, delegated*) $\preccurlyeq$ (*read, delegated*), (*read, original*) $\preccurlyeq$ (*read, delegated*) and (*idle, original*) $\preccurlyeq$ (*written, delegated*). Figure 6(a) represents the graph of the risk ordering relation, while Figure 6(b) represents the equivalent banded risk graph. Possibly due to states (*written, original*) and (*written, delegated*) not receiving sufficient attention in the risk assessment process, they are shown to be non–comparable by the $\sqsubseteq$ relation with many other states. As a precaution against a possible inadequacy in the risk assessment process, they have been placed conservatively in the highest risk band of the risk graph. This could be seen as a flag, alerting the designer to reconsider the risk levels of these states if the circumstances do not warrant such an interpretation.

As was mentioned in Section II-E, Safecharts disallows transitions between *risk-noncomparable* states. Though they may appear, according to the AND state shown in Figure 5(b), transitions between risk-noncomparable states, such as those between (*written, original*) and (*written, delegated*), which are not permitted by the risk graph in the security layer. Moreover, because of the positions of their source and target states in the risk graph, transitions (*idle, original*)$\rightsquigarrow$(*written, delegated*) and (*idle, original*)$\rightsquigarrow$(*read, delegated*) 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 secure nondeterminism. If this is unacceptable on security grounds, then the designer needs to verify the security policy and/or the relative risk levels of the two target states, namely (*written, delegated*) and (*read, delegated*). It may also be the case that no roles are to be delegated while in the middle of higher risk operation. This may be a case to be borne in mind when including pairs such as (*read, original*) and (*read, delegated*) in the $\sqsubseteq$ relation, thus explicitly sanctioning, or barring, transitions between them relying on built-in rules of Safecharts. An exception for this would be a situation involving training, where a delegation of a role in the middle of an operation may be justifiable, but even in this case, consistency across the operations may need to be observed so that the delegation concerns the particular operation being exercised by the delegating subject and not any arbitrary operation. Though this situation has been precipitated by the manner of derivation of banded risk graphs from the risk ordering relation $\sqsubseteq$, this might be another instance where the designer needs to verify the appropriateness of the security policy being followed. Thus, the risk graph is not merely a form of representation of risk but also a means of refining the risk assessment process itself.

V. Conclusion

The objective of this paper has been to introduce a novel use of Safecharts in the specification and modelling of security requirements. Safecharts was originally developed as a safety–oriented variant of Statecharts explicitly for safety–critical systems design. Nevertheless, 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 access rights in data security as understood in RBAC. In this respect, a state–based model of delegation in RBAC allows the evaluation of the current state of access rights in granting, or denying, such rights based on security considerations and the current assignment of roles. Various features of Safecharts, such as its unique concept of risk graph capturing the risks posed by different states, the security–driven enforcement of transitions and the imposition of a conservative security–oriented default assumptions on security risks of states, have been put to use in modelling security requirements and evaluating their effectiveness, as well as, in enforcing some desirable properties such as integrity and confidentiality. As was the case in the safety domain, built in mechanisms and assumptions are designed to prompt the designer to question their validity in the context of the specific application being dealt with, thereby enhancing the overall security of the system.

Correct interpretation of Safecharts, both in the context of security and safety, requires a sound understanding of several important aspects of its semantics. Prominent among them are the risk graph of AND states and the definition of its step semantics. These new developments have been introduced in this paper. On–going and future research is aimed at modelling more complex security models in Safecharts and the extension of Safecharts so that it can serve as a single unified framework for the specification and modelling of dependability properties, such as safety and security, both in isolation or in any form of combination.

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

[3] 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, 1999.

[4] Glinz, M. *An Integrated Formal Model of Scenarios Based on Statecharts.* Proceedings of the 5th European Software Engineering Conference, Sitges, Spain. Berlin, etc.: Springer, LNCS(989), pp. 254-271, 1995.

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

[6] Harel D. and Naamad A. *The STATEMATE semantics of Statecharts.* ACM Transactions on Software Engineering and Methodology, Vol. 5(4), pp. 293–333, October 1996.

[7] Huizing C. and Roever W. *Introduction to Design Choices in the Semantics of Statecharts.* Information Processing Letters, Vol. 37(4), pp. 205–213, 1991.

[8] Leveson N. *Safeware, System Safety and Computers.* Addison–Wesley, 1995.

[9] 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, 2003.

[10] Nissanke N. and Dammag .H *Design for safety in Safecharts with risk ordering of states.* Safety Science, Vol. 40, pp. 753–763, 2002.

[11] Pnueli A. and Shalev A. *What is in a Step: On the Semantics of Statecharts.* Theoretical Aspects of Computer Software, LNCS 526, pp. 244–265, Springer, 1991.

[12] Rushby J. *Kernels for Safety*? Safe and Secure Computing systems, T. Anderson. editor, Chapter 13, pp. 210–220, Blackwell Scientifc Publications, 1989.

[13] Rushby J. *Critical System Properties: Survey and Taxonomy.* Reliability Engineering and System Safety, Vol. 43 (2), pp. 189–219, 1994.

[14] Sandhu R., Ferraiolo D. and Kuhn K. *The NIST Model for Role-Based Access Control: Towards A Unified Standard.* Proceedings of the 5th ACM workshop on Role-Based Access Control, pp. 47–63, Berlin, Germany, June, 2000.

[15] Simpson A., Woodcock J. and Davies J. *Safety through security.* Proceedings of the 9th International Workshop on Software Specification and Design, pp. 18–24, 1998.

[16] Stavridou V. and Dutertre B. *From Security to Safety and Back.* Proceedings of Computer Security, Dependability and Assurance, pp. 182–195, 1999.