Self-Adaptive Role-Based Access Control for Business Processes

Carlos Eduardo da Silva*, José Diego Saraiva da Silva†, Colin Paterson‡ and Radu Calinescu‡
*Metropole Digital Institute (IMD), Federal University of Rio Grande do Norte, Natal, RN, Brazil
†Federal Institute of Education, Science and Technology of Rio Grande do Norte - IFRN, Natal, RN, Brazil
‡Department of Computer Science, University of York, York, UK

***Abstract*—We present an approach for dynamically reconfiguring the role-based access control (RBAC) of information systems running business processes, to protect them against insider threats. The new approach uses business process execution traces and stochastic model checking to establish confidence intervals for key measurable attributes of user behaviour, and thus to identify and adaptively demote users who misuse their access permissions maliciously or accidentally. We implemented and evaluated the approach and its policy specification formalism for a real IT support business process, showing their ability to express and apply a broad range of self-adaptive RBAC policies.**

I. Introduction

Security incidents caused by insider threats may result in severe financial and reputational loss [1][2]. Insider threats arise when trusted users of an information system can exploit their access permissions to compromise the confidentiality, integrity or availability of an organisation's information assets [3], [4]. These trusted users include employees, contractors and business partners who can cause harm intentionally (e.g. for personal gain or revenge) or through error (e.g. due to negligence or insufficient training) [5].

To mitigate these threats, information systems employ control mechanisms that restrict the access to their assets. More often than not, these mechanisms implement the *role-based access control* (RBAC) [6] model, where the permissions to execute operations on information assets are associated with *roles*, and the users are only assigned the role(s) they need to perform their jobs. As such, RBAC restricts user access, and helps detect access violation attempts. However, it cannot detect users who maliciously or accidentally abuse their legitimate access permissions. Furthermore, it is unable to mitigate such abuse [7], even when a separate insider attack detection mechanism [8] is available. For example, in the context of a ticket support system, a support attendant with an elevated number of tickets opened on behalf of clients that are abandoned can be considered an anomaly not detectable by RBAC.

Our work addresses this limitation of traditional RBAC in the context of business processes. To this end, we exploit activity logs already available for many important businesses processes, which also enable the monitoring of the activities undertaken by individual users of these processes. Using this information, dynamic access control mechanisms can be employed to respond to abnormal user behaviour through actions decided based on risk analysis and pre-defined *adaptation policies*. Such actions may include changes to authorisation policies, modifications of user assignment to roles and of role permissions, user training, and changes to the business process.

In this paper, we introduce a self-adaptive RBAC (saRBAC) approach that enacts these general principles by dynamically reconfiguring user assignments to roles in order to mitigate insider threats. As an example, users with abnormal behaviour may be removed from a role or may be demoted to roles with restricted permissions. Our saRBAC approach is underpinned by the analysis of stochastic models that enable the comparison of individual user behaviour to the average behaviour of the other users in the same role. Given a business process and traces of its execution obtained through monitoring, saRBAC (a) builds a parametric Markov model of the process, and (b) uses FACT [9], [10], a probabilistic model checker, to establish confidence intervals for model properties associated with key aspects of user behaviour. For each user and analysed property, two confidence intervals are computed corresponding to the property value for the user, and for all the other users taken together, respectively. If the two confidence intervals do not overlap, then saRBAC concludes that the examined user behaves (statistically) differently from the other users. The analysed properties, the definition of what constitutes abnormal behaviour, and the actions required when such behaviour is detected are formally specified in saRBAC adaptation policies.

We evaluated saRBAC within the information system running the IT support business process at the Federal Institute of Education, Science, and Technology of Rio Grande do Norte (IFRN), Brazil, an organisation with over 44,000 users. We devised the saRBAC adaptation policies together with the IFRN management team. Because of the business-critical nature of the system, we ran saRBAC as an advisory system suggesting access control modifications that IT managers could verify instead of implementing them directly. However, the code for a fully automated operation is in place, and given the positive evaluation results (described later in the paper) we expect it to be activated within the near future.

The rest of the paper is organised as follows. Section II presents background information on stochastic modelling and probabilistic model checking with confidence intervals. Section III introduces the real case study used to illustrate and evaluate our approach. Section IV describes the saRBAC

approach and its adaptation policies. The saRBAC implementation we used for the IFRN system and the evaluation results are presented in Sections V and VI, respectively. Finally, Section VII compares our approach with related research, and Section VIII concludes the paper with a brief summary.

II. Preliminaries

Discrete-time Markov chains (DTMCs) DTMCs provide a formal modelling framework for state transition systems in which the selection of successor states is controlled through probabilistic choice [11]. A Markov chain is memoryless, in that the next state only relies on the current state and not the path that led to the current state.

Definition 1. A (labeled) DTMC over an atomic proposition set AP is a tuple

$$\mathcal{M} = (S, s_0, \boldsymbol{P}, L) \tag{1}$$

where S is a finite set of states; $s_0 \in S$ is the initial state; $\boldsymbol{P} : S \times S \to [0,1]$ is the transition probability function such that for each state s the probability of moving to state s' in a single transition is $\boldsymbol{P}(s,s')$ and $\sum_{s' \in S} \boldsymbol{P}(s,s') = 1$; and $L : S \to 2^{AP}$ is a labelling function that maps each state to the set of atomic propositions that hold in that state.

A *path* π over $\mathcal{M}$ is a possibly infinite sequence of states from S such that for any adjacent states s and s' in π, $\mathbf{P}(s,s') > 0$. The m-th state on a path π, $m \geq 1$, is denoted $\pi(m)$. Finally, for any state $s \in S$, $Paths^{\mathcal{M}}(s)$ represents the set of all infinite paths over $\mathcal{M}$ that start with state s.

In using DTMCs to model real-world systems, it is often the case that the DTMC states and transitions can be derived (sometimes automatically) from existing system artefacts such as activity diagrams, architectural models or, in the case of software, from the actual code. In contrast, the probabilities associated with the state transitions are more difficult to determine, requiring the use of *parametric DTMCs*.

Definition 2. A parametric DTMC is a discrete-time Markov chain (1) in which some or all the transition probabilities $\boldsymbol{P}$ are unknown.

Cost/reward structures are used to extend the range of properties that can be verified using DTMCs. These structures associate nonnegative values with the states and/or transitions of a (parametric) DTMC. Depending on the verified property, these values are interpreted as costs (e.g. resource use or price) or rewards (e.g. throughput or profit).

Definition 3. A *cost/reward structure* over a Markov chain $\mathcal{M} = (S, s_0, \mathbf{P}, L)$ is a pair of functions $(\underline{\rho}, \iota)$ such that:

- $\underline{\rho} : S \to \mathbb{R}_{\geq 0}$ is the *state reward function* (a vector);
- $\iota : S \times S \to \mathbb{R}_{\geq 0}$ is the *transition reward function* (a matrix).

Probabilistic computation tree logic (PCTL) PCTL [12] provides a formal language for the specification of verifiable properties of DTMCs.

Definition 4. Let AP be a set of atomic propositions and $a \in AP$, $p \in [0,1]$, $k \in \mathbb{N}$, $r \in \mathbb{R}$ and $\bowtie \in \{\geq, >, <, \leq\}$. Then a *state formula* Φ and a *path formula* Ψ in probabilistic computation tree logic (PCTL) are defined by the grammar:

$$\Phi ::= true \mid a \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{P}_{\bowtie p}[\Psi] \tag{2}$$

$$\Psi ::= X\Phi \mid \Phi \mathrm{U} \Phi \mid \Phi \mathrm{U}^{\leq k} \Phi \tag{3}$$

and a *cost/reward state formula* is defined by the grammar:

$$\Phi ::= \mathcal{R}_{\bowtie r}[\mathrm{I}^{=k}] \mid \mathcal{R}_{\bowtie r}[\mathrm{C}^{\leq k}] \mid \mathcal{R}_{\bowtie r}[\mathrm{F}\Phi] \mid \mathcal{R}_{\bowtie r}[\mathrm{S}]. \tag{4}$$

The semantics of PCTL is defined with a satisfaction relation $\models$ over the states S and the paths $Paths^{\mathcal{M}}(s)$, $s \in S$, of a DTMC (1). Thus, $s \models \Phi$ means "Φ is satisfied in state s". For any state $s \in S$, we have: $s \models true$; $s \models a$ iff $a \in L(s)$; $s \models \neg\Phi$ iff $\neg(s \models \Phi)$; and $s \models \Phi_1 \wedge \Phi_2$ iff $s \models \Phi_1$ and $s \models \Phi_2$. A state formula $\mathcal{P}_{\bowtie p}[\Psi]$ is satisfied in a state s if the probability of the future evolution of the system satisfying Ψ satisfies $\bowtie p$, where:

- the "next" formula $X\Phi$ is satisfied by a path π iff Φ is satisfied in the next state of π (i.e., in state $\pi(2)$);
- the time bounded "until" formula $\Phi_1 \mathrm{U}^{\leq k} \Phi_2$ is satisfied by a path π iff Φ_1 is satisfied in each of the first x states of π for some $x < k$, and Φ_2 is satisfied in the $(x+1)$-th state of π;
- the unbounded "until" formula $\Phi_1 \mathrm{U} \Phi_2$ is satisfied by a path π iff Φ_1 is true in each of the first $x > 0$ states of π, and Φ_2 is true in the $(x+1)$-th state of π.

The notation $\mathrm{F}^{\leq k}\Phi \equiv true \mathrm{U}^{\leq k}\Phi$ and $\mathrm{F}\Phi \equiv true \mathrm{U}\Phi$ is used when the first part of a bounded until and until formula, respectively, is $true$.

In addition, given a cost/reward structure in the form from Definition 3, PCTL was extended with *reward constraints* that support the specification of both expected and cumulative rewards [13]. Thus, the cost/reward operator $\mathcal{R}$ can be used to analyse the expected cost at timestep k ($\mathcal{R}_{\bowtie r}[\mathrm{I}^{=k}]$), the expected cumulative cost up to time step k ($\mathcal{R}_{\bowtie r}[\mathrm{C}^{\leq k}]$), the expected cumulative cost to reach a future state that satisfies a property Φ ($\mathcal{R}_{\bowtie r}[\mathrm{F}\Phi]$), and the expected steady-state reward in the long run ($\mathcal{R}_{\bowtie r}[\mathrm{S}]$).

Probabilistic model checking with confidence intervals *Probabilistic model checkers* (e.g. PRISM [14] and MRMC [15]) use symbolic model checking algorithms to establish if a PCTL formula $\mathcal{P}_{\bowtie p}[\Psi]$ is satisfied by calculating the actual probability that Ψ is satisfied, and comparing it with the bound p. Therefore, calculating the actual probability does not add any complexity, and the extended PCTL syntax $\mathcal{P}_{=?}[\Psi]$ can be used (for the outermost $\mathcal{P}$ operator of a PCTL formula) to obtain this probability. This also applies to cost/reward PCTL formulae, for which $\mathcal{R}_{=?}[\mathrm{I}^{=k}]$, $\mathcal{R}_{=?}[\mathrm{C}^{\leq k}]$, $\mathcal{R}_{=?}[\mathrm{F}\Phi]$, and $\mathcal{R}_{=?}[\mathrm{S}]$ are used similarly.

While "standard" probabilistic model checking has been used to develop self-adaptive systems before (e.g. [16], [17], [18], [19]), in this paper we use the recently introduced probabilistic model checker FACT [9], [10], which computes *confidence intervals* for the extended-syntax properties $\mathcal{P}_{=?}[\Psi]$ and $\mathcal{R}_{=?}[\ldots]$ of parametric DTMCs for which observations

of the state transitions associated with unknown probabilities are available. Notice that this is completely different from the established practice of assuming that the transition probabilities have fixed known values and using discrete-event simulation (sometimes called "statistical model checking") to compute confidence intervals for the properties of non-parametric DTMCs. This functionally is provided by many model checkers, including PRISM. In contrast, FACT takes into account the fact that transition probabilities are unknown multinomial-distributed random variables, uses precise parametric model checking to obtain a closed-form expression of the analysed property, and exploits actual observations of the transitions to calculate a confidence interval for this expression. As such, FACT is particularly suited for our approach, where business processes logs containing such observations are available but the probabilities with which different activities are executed are typically unknown.

III. Case Study

We will illustrate and evaluate our approach using a business process implemented by the SUAP[1] information system of the Federal Institute of Education, Science, and Technology of Rio Grande do Norte (IFRN), Brazil. SUAP is used for most of IFRN's administrative processes, is actively developed and maintained by the IT Management Directorate of IFRN, and has been adopted by 25 federal institutes all over Brazil. The IFRN SUAP users comprise 12,500 academic, administrative, technical staff and contractors, and 32,000 students, located at the 26 IFRN campuses in the state of Rio Grande do Norte.

The SUAP business process we use in our work is the *Ticket Support* process from Figure 1, which allows IFRN users to request IT support services (e.g. access and password changes) and to report IT-related problems. SUAP's business processes are security sensitive, and the RBAC access control model is used to enforce security policies. In this context, the Ticket Support process involves three roles: *Client*, *Support* and *Administrator*. A *Client* is any user who needs some IT service done, and can raise issues by opening tickets. A *Support* user is an employee responsible for dealing with tickets, e.g. an IT technician or analyst. The system also includes the *Administrator* role, which comprises users responsible for supervising the work carried out by *Support* users.

As shown in Figure 1 the Ticket Support process starts with the opening of a ticket, either by a client (*Open ticket*) or by a support attendant on behalf of a client (*Open ticket of behalf*), e.g., the client might go to the IT department in person to raise an issue. An open ticket can be cancelled by the client (*Cancel ticket by user*), e.g., if the ticket was opened by mistake, or can be allocated to a support attendant. Ticket allocation can be done by an administrator (*Allocate to support*), or by support attendants themselves (*Allocate to self*).

Once allocated, a support attendant will work on the ticket (*Check ticket*)—solving the issue (*Solve ticket*), reallocating the ticket to a different support attendant (*Reallocate ticket*), cancelling the ticket (*Cancel ticket by support*), or suspending the ticket and asking the client to provide more information about the issue (*Suspend ticket*). Suspended tickets are sent back to clients, who can either reply to the support attendant (*Add more information*), or cancel the ticket (*Cancel ticket by user*). If the client does not reply within a specific time, the ticket is considered abandoned and is cancelled by the support attendant (*Cancel abandoned ticket*).

A solved ticket is sent back to its client, who can confirm the resolution of the issue by closing it (*Close ticket*), or reopen the ticket (*Reopen ticket*) indicating that the issue is not resolved. A solved ticket not handled by the client within a certain time frame is closed by an administrator (*Close expired ticket*).

We used SUAP's extensive logging capabilities to obtain detailed execution traces of the Ticket Support process for the three-month period between May–July 2016, and we interviewed the IFRN management team to determine the organisation's insider-threat concerns for the process. As such, we learnt about concerns based both on past cases of internal abuse and on yet unconfirmed incident scenarios identified by their security risk management procedures. Using these concerns, we defined a set of adaptation policies that capture abnormal behaviours of Ticket Support users, and preferred ways of dealing with them. Table I shows a representative subset of these policies, expressed informally in terms of confidence intervals for measurable attributes of user behaviour. The appropriate confidence levels for these policies were initially unknown, so they were obtained through experimental calibration such as to minimise the number of false positives and false negative over a subset of users whose behaviour was carefully examined by IT managers. Our saRBAC approach formalises these adaptation policies and implements them as described in the next section.

IV. The saRBAC Approach

A. Business process description

Our approach is applicable to a business process implemented by an RBAC-based information system whose users are organised into $n>0$ roles. We assume that the sets of users associated with the n roles are $Role_1$, $Role_2$, …, $Role_n$ (so the set of all users is $Users = \bigcup_{i=1}^{n} Role_i$), that the business process carries out activities from an activity set A, and that a UML activity diagram of the process is available. Also, we use $A_i \subseteq A$ to denote the set of activities that can be executed by $Role_i$, $1 \leq i \leq n$. Accordingly, the set of activities that a generic user $u \in Users$ has permissions to execute is $perm(u) = \{a \in A \mid \exists 1 \leq i \leq n \bullet u \in Role_i \wedge a \in A_i\}$.

To augment the business process with *self-adaptive* RBAC capabilities, we require that monitoring is used to record traces of all process executions, and assume that traces have the form

$$\langle a_1, u_1, a_2, u_2, \ldots, a_m, u_m \rangle, \tag{5}$$

where $a_1, a_2, \ldots, a_m \in A$ is the ordered list of activities performed during an execution of the business process, and $u_1, u_2, \ldots, u_m \in Users$ are the users that carried out each

[1] *Sistema Unificado de Administrao Pública* – Unified System for Public Administration

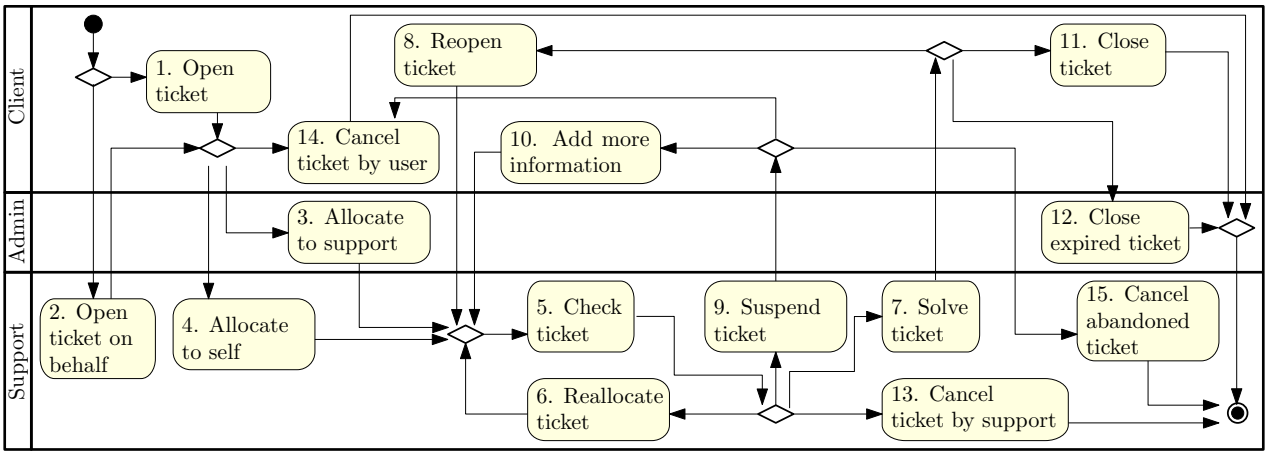

Fig. 1. Ticket Support business process (UML activity diagram produced during IFRN's development of the SUAP information system)

TABLE I
ADAPTATION POLICIES FOR THE TICKET SUPPORT PROCESS

ID	Description
P1	A client whose expected number of reopens per ticket (at *confidence level* $\alpha = 0.95$) is larger than that of the other clients should be not be allowed to open new tickets.
P2	A support attendant whose expected number of suspensions per ticket ($\alpha = 0.95$) exceeds that of the other attendants should be closely monitored and should need approval for suspending tickets. Additionally, the clients that have been served by this attendant should be considered; if their expected number of suspensions per ticket *for tickets handled by other attendants* ($\alpha = 0.99$) is lower than that of the other clients, then the investigated attendant should be suspended.
P3	A support attendant whose probability of cancelling ticket that have not been suspended ($\alpha = 0.95$) exceeds that of the other support team members should need approval for cancelling tickets.
P4	A support attendant whose expected number of tickets opened on behalf of clients are abandoned ($\alpha = 0.95$) exceeds that of the other support team members should no longer be allowed to open tickets on behalf of clients.
P5	A support attendant whose expected number of reallocations, suspensions, cancellations or reopens per ticket ($\alpha = 0.80$) exceeds that of the other support attendants should be placed under observation. Additionally, if this discrepancy is also confirmed with a higher confidence level of $\alpha = 0.95$, the support attendant should need approval for his or her actions.
P6	A client whose expected number of suspensions, reopens, abandonments or cancellations by support per ticket ($\alpha = 0.90$) exceeds that of the other clients should be placed under observation. Additionally, the support attendants that have dealt with by this client should be considered; if their expected number of suspensions, reopens, abandonments or cancellations by support per ticket ($\alpha = 0.90$) is lower than that of the other support attendants, then the investigated client should not be allowed to open new tickets.

of these activities, respectively. We use T to denote the set of all recorded business process traces, and $traces(u) \subseteq T$ to represent the set of all traces comprising at least one activity performed by user $u \in Users$:

$$traces(u) = \{\langle a_1, u_1, \ldots, a_m, u_m\rangle \in T \mid \exists 1 \leq i \leq m \bullet u_i = u\}.$$

Finally, the set of traces for a user subset $U \subseteq Users$ is given by $traces(U) = \bigcup_{u \in U} traces(u)$.

Example 1. The Ticket Support process from Section III uses $n = 3$ roles, i.e. $Role_1 = Client$, $Role_2 = Support$ and $Role_3 = Admin$. The activity sets for these roles are $A_1 = \{\mathsf{Open}, \mathsf{Reopen}, \mathsf{AddInformation}, \mathsf{Close}, \mathsf{CancelByUser}\}$, $A_2 = \{\mathsf{OpenOnBehalf}, \mathsf{AllocateToSelf}, \mathsf{Check}, \mathsf{Reallocate}, \mathsf{Solve}, \mathsf{Suspend}, \mathsf{CancelBySupport}, \mathsf{CancelAbandoned}\}$, $A_3 = \{\mathsf{AllocateToSupport}, \mathsf{CloseExpired}\}$, and the entire activity set is $A = A_1 \cup A_2 \cup A_3$. A possible Ticket Support execution trace involving users $u_1 \in Client$ and $u_2 \in Support$ is $\langle \mathsf{Open}, u_1, \mathsf{AllocateToSelf}, u_2, \mathsf{Check}, u_2, \mathsf{Solve}, u_2, \mathsf{Close}, u_1\rangle$.

B. Parametric DTMC of a business process

For a business process with the above characteristics, we devise a parametric DTMC model (1) from its UML activity diagram as follows. First, we build the finite state set S comprising a state for each activity node from the diagram, and an initial state s_0 and a final state s_F for the initial and final nodes of the activity diagram, respectively; let $node(s)$ denote the activity diagram node associated with state $s \in S$. We then assemble the transition probability matrix $\boldsymbol{P}$:

- We set $\boldsymbol{P}(s, s') = 1.0$ for every pair of states $s, s' \in S$ for which the node reached immediately after $node(s)$ (i.e. without going through intermediate nodes associated with states from S) is *always* $node(s')$, and for $s = s' = s_\mathsf{F}$.
- We associate an unknown transition probability $\boldsymbol{P}(s, s')$ with each pair of states s, s' for which $node(s')$ can be reached from $node(s)$ going only through decision nodes in the activity diagram.
- We set $\boldsymbol{P}(s, s') = 0$ for every other pair of states $s, s' \in S$.

Next, we define a labelling function L that labels each state $s \in S$ with an atomic proposition that suggestively reflects the state of the system after the execution of the activity associated with $node(s)$. For instance, we used the label $Allocated$ for the DTMC state associated with the activity $\mathsf{AllocateToSupport}$ of our Ticket Support process. Finally, we augment the resulting DTMC with cost/reward structures modelling the measurable attributes of user behaviour from a set of adaptation policies similar to those from Table I.

Example 2. Applying the DTMC derivation method described

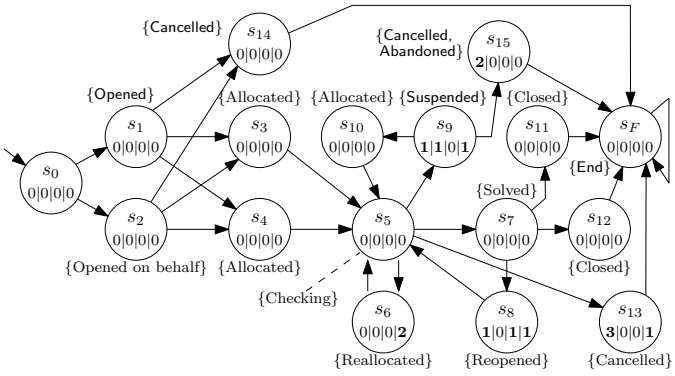

Fig. 2. Parametric DTMC model of the Ticket System. The transition probabilities are not shown because they are user-dependent and unknown (for states with multiple outgoing transitions) or 1.0 (for states with a single outgoing transition).

above to the Ticket Support activity diagram from Figure 1 yields the DTMC shown in Figure 2. The 17 states of this DTMC correspond to the 15 activities of the business process plus the initial and final states of the activity diagram, and the labelling captures the state of a ticket during the execution of the process. The number combinations $r_1|r_2|r_3|r_4$ annotating the DTMC states define four cost/reward structures (cf. Definition 1) used to formalise the policies from Table I: an "expensive" client structure (r_1, cf. policy P6 from Table I), a "suspended" structure (r_2, cf. policy P2), a "reopened" structure (r_3, cf. policy P1), and a "lazy" support staff structure (r_4, cf. policy P5). For example, state s_6 (which corresponds to a ticket being reallocated by a member of the support team) is annotated with $r_1|r_2|r_3|r_4 = 0|0|0|2$, indicating that ticket reallocation is not characteristic of an expensive client ($r_1 = 0$), is not a reopen or suspension ($r_2 = r_3 = 0$), but is a strong characteristic of "lazy" support staff ($r_4 = 2$).

C. Self-adaptive RBAC policies

Given a business process, its parametric DTMC $\mathcal{M} = (S, S_0, \boldsymbol{P}, AP, L)$ constructed as described in Section IV-B, and its set of execution traces T, the self-adaptation policies supported by our approach are sequences of rules

$$policy = \langle rule_1, rule_2, \ldots, rule_N \rangle, \tag{6}$$

where each rule $rule_i$, $1 \leq i \leq N$, is a tuple

$$rule_i = (\mathit{filter}_i, \Phi_i, \bowtie_i, \alpha_i, post_i). \tag{7}$$

The elements of the above rule are as follows:

1) filter_i is a first-order logic expression that specifies constraints for $2n$ user sets $R^{\text{test}}_{i,1}, R^{\text{ref}}_{i,1} \subseteq Role_1$, $R^{\text{test}}_{i,2}, R^{\text{ref}}_{i,2} \subseteq Role_2$, …, $R^{\text{test}}_{i,n}, R^{\text{ref}}_{i,n} \subseteq Role_n$. For example, the filter

$$R^{\text{test}}_{i,1} = \{u\} \ \wedge \ R^{\text{ref}}_{i,1} = Role_1 \setminus R^{\text{test}}_{i,1}$$

holds for all pairs of user sets $(R^{\text{test}}_{i,1}, R^{\text{ref}}_{i,1})$ for which $R^{\text{test}}_{i,1}$ consists of a single user $u \in Role_1$ and $R^{\text{ref}}_{i,1}$ comprises all the users from $Role_1$ except user u.
2) ϕ_i is a PCTL formula over the atomic propositions AP.
3) $\bowtie_i \in \{<, \leq, \geq, >\}$ is a relational operator;
4) $\alpha_i \in (0, 1)$ is a confidence level.
5) The *postcondition* $post_i$ is a first-order logic expression which specifies constraints that must hold if $rule_i$ is applied as explained below. For example, the postcondition $\forall u \in R^{\text{test}}_{i,1} \bullet perm'(u) = perm(u) \setminus \{\mathsf{Open}\}$ holds if the users in $R^{\text{test}}_{i,1}$ are no longer able to execute the operation Open after the application of $rule_i$.

Algorithm 1 Application of saRBAC rule and policy

```
1:  function ApplyRule(M, policy, i, Λ)
2:      for all R^test_{i,1}, R^ref_{i,1} ⊆ Role_1, R^test_{i,2}, R^ref_{i,2} ⊆ Role_n, …,
                R^test_{i,n}, R^ref_{i,n} ⊆ Role_n such that filter_i holds do
3:          T^test_i ← ⋂_{1≤j≤n, R^test_{i,j}≠∅} traces(R^test_{i,j})
4:          T^ref_i ← ⋂_{1≤j≤n, R^ref_{i,j}≠∅} traces(R^ref_{i,j})
5:          [a^test_i, b^test_i] ← ConfInterval(M, T^test_i, Φ_i, α_i)
6:          [a^ref_i, b^ref_i] ← ConfInterval(M, T^ref_i, Φ_i, α_i)
7:          if [a^test_i, b^test_i] ⋈_i [a^ref_i, b^ref_i] then
8:              ReconfigureRBAC(post_i)
9:              if i < N then
10:                 Λ ← Λ⌢⟨R^test_{i,1}, R^test_{i,2}, …, R^test_{i,n}⟩
11:                 ApplyRule(M, policy, i+1, Λ)
12:             end if
13:         end if
14:     end for
15: end function
16:
17: function ApplyPolicy(M, policy)
18:     ApplyRule(M, policy, 1, ⟨⟩)
19: end function
```

The application of rule (7) involves the execution of function ApplyRule from Algorithm 1. This function takes as parameters the DTMC model $\mathcal{M}$ of the business process, a *policy* (6), the index i of the rule to apply, and the sequence $\Lambda = \langle R^{\text{test}}_{1,1}, R^{\text{test}}_{1,2}, \ldots, R^{\text{test}}_{1,n}, \ldots, R^{\text{test}}_{i-1,1}, R^{\text{test}}_{i-1,2}, \ldots, R^{\text{test}}_{i-1,n} \rangle$ of "test" user sets from the applications of the previous $i-1$ rules of *policy*. The function executes the for loop in lines 2–14 over all distinct combinations of user sets $R^{\text{test}}_{i,1}, R^{\text{ref}}_{i,1}$, $R^{\text{test}}_{i,2}, R^{\text{ref}}_{i,2}$, …, $R^{\text{test}}_{i,n}, R^{\text{ref}}_{i,n}$ that satisfy filter_i. For each such combination, subsets of *test traces* T^{test}_i and *reference traces* T^{ref}_i are first built in lines 3 and 4. These subsets contain the traces of all business process executions that comprise at least one activity carried out by a user from each of the non-empty user sets $R^{\text{test}}_{i,1}$, $R^{\text{test}}_{i,2}$, …, $R^{\text{test}}_{i,n}$ and $R^{\text{ref}}_{i,1}$, $R^{\text{ref}}_{i,2}$, …, $R^{\text{ref}}_{i,n}$, respectively. Next, the parametric DTMC $\mathcal{M}$ and the observations from the trace subsets T^{test}_i and T^{ref}_i are used to establish test and reference confidence intervals $[a^{\text{test}}_i, b^{\text{test}}_i]$ and $[a^{\text{ref}}_i, b^{\text{ref}}_i]$ for the property Φ_i, with confidence level α_i (lines 5 and 6). Finally, if the rule *precondition* $[a^{\text{test}}_i, b^{\text{test}}_i] \bowtie_i [a^{\text{ref}}_i, b^{\text{ref}}_i]$[2] (line 7) then (i) the RBAC system is dynamically reconfigured (as described in the next section) so that $post_i$ holds (line 8); and (ii) if $rule_i$ is not the last rule from policy (6) (i.e. if $i < N$), $rule_{i+1}$ is also applied (lines

[2]We extend the relational operators to work with intervals in the natural way, e.g. $[x_1, y_1] < [x_2, y_2]$ iff $y_1 < x_2$.

TABLE II
FORMALISED SELF-ADAPTATION POLICIES FOR THE TICKET SUPPORT BUSINESS PROCESS

ID	i	$filter_i$	Φ_i	$\bowtie_i$	α_i	$post_i$
P1	1	$R^{\text{test}}_{1,1}=\{u\} \wedge R^{\text{test}}_{1,2}=Support \wedge$ $R^{\text{ref}}_{1,1}=Client\setminus\{u\} \wedge R^{\text{ref}}_{1,2}=Support$	$\mathcal{R}^{\text{reopened}}_{=?}[\text{F End}]$	$>$	.95	$perm'(u)=perm(u)\setminus\{\text{Open}\}$
P2	1	$R^{\text{test}}_{1,1}=Client \wedge R^{\text{test}}_{1,2}=\{u\} \wedge$ $R^{\text{ref}}_{1,1}=Client \wedge R^{\text{ref}}_{1,2}=Support\setminus\{u\}$	$\mathcal{R}^{\text{suspended}}_{=?}[\text{F End}]$	$>$	.95	$perm'(u)=(perm(u)\setminus\{\text{Suspend}\})\cup$ $\{\text{SuspendWithApproval}\}$
	2	$R^{\text{test}}_{2,1}=\{v\in Client \mid traces(v)\cap traces(R^{\text{test}}_{1,2})\neq\emptyset\}$ $\wedge R^{\text{test}}_{2,2}=Support\setminus R^{\text{test}}_{1,2} \wedge$ $R^{\text{ref}}_{2,1}=Client\setminus R^{\text{test}}_{2,1} \wedge R^{\text{ref}}_{2,2}=R^{\text{test}}_{2,2}$	$\mathcal{R}^{\text{suspended}}_{=?}[\text{F End}]$	$<$	.99	$u\in R^{\text{test}}_{1,2} \rightarrow perm'(u)=\emptyset$
P3	1	$R^{\text{test}}_{1,1}=Client \wedge R^{\text{test}}_{1,2}=\{u\} \wedge$ $R^{\text{ref}}_{1,1}=Client \wedge R^{\text{ref}}_{1,2}=Support\setminus\{u\}$	$\mathcal{P}_{=?}[\neg\text{Suspended U Cancelled}]$	$>$	.95	$perm'(u)=(perm(u)\setminus\{\text{Cancel}\})\cup$ $\{\text{CancelWithApproval}\}$
P4	1	$R^{\text{test}}_{1,1}=Client \wedge R^{\text{test}}_{1,2}=\{u\} \wedge$ $R^{\text{ref}}_{1,1}=Client \wedge R^{\text{ref}}_{1,2}=Support\setminus\{u\}$	$\mathcal{P}_{=?}[\neg\text{Opened U Abandoned}]$	$>$	.95	$perm'(u)=perm(u)\setminus\{\text{OpenOnBehalf}\}$
P5	1	$R^{\text{test}}_{1,1}=Client \wedge R^{\text{test}}_{1,2}=\{u\} \wedge$ $R^{\text{ref}}_{1,1}=Client \wedge R^{\text{ref}}_{1,2}=Support\setminus\{u\}$	$\mathcal{R}^{\text{lazy}}_{=?}[\text{F End}]$	$>$	.80	$perm'(u)=(perm(u)\setminus\{\text{Check},$ $\text{Solve, Suspend}\})\cup\{\text{MonitoredCheck},$ $\text{MonitoredSolve, MonitoredSuspend}\}$
	2	$R^{\text{test}}_{2,1}=Client \wedge R^{\text{test}}_{2,2}=R^{\text{test}}_{1,2} \wedge$ $R^{\text{ref}}_{2,1}=Client \wedge R^{\text{ref}}_{2,2}=R^{\text{ref}}_{1,2}$	$\mathcal{R}^{\text{lazy}}_{=?}[\text{F End}]$	$>$	.95	$perm'(u)=(perm(u)\setminus\{\text{Check},$ $\text{Solve, Suspend}\})\cup\{\text{CheckWithApproval},$ $\text{SolveWithApproval, SuspendWithApproval}\}$
P6	1	$R^{\text{test}}_{1,1}=\{u\} \wedge R^{\text{test}}_{1,2}=Support \wedge$ $R^{\text{ref}}_{1,1}=Client\setminus\{u\} \wedge R^{\text{ref}}_{1,2}=Support$	$\mathcal{R}^{\text{expensive}}_{=?}[\text{F End}]$	$>$	.90	$perm'(u)=(perm(u)\setminus\{\text{Open},$ $\text{Reopen,AddInformation}\})\cup\{\text{MonitoredOpen},$ $\text{MonitoredReopen,MonitoredAddInformation}\}$
	2	$R^{\text{test}}_{2,1}=Client\setminus R^{\text{test}}_{1,1} \wedge R^{\text{test}}_{2,2}=\{v\in Support \mid$ $traces(v)\cap traces(R^{\text{test}}_{1,1})\neq\emptyset\} \wedge$ $R^{\text{ref}}_{2,1}=R^{\text{test}}_{2,1} \wedge R^{\text{ref}}_{2,2}=Support\setminus R^{\text{test}}_{2,2}$	$\mathcal{R}^{\text{expensive}}_{=?}[\text{F End}]$	$<$	.90	$u\in R^{\text{test}}_{1,1} \rightarrow perm'(u)=perm(u)\setminus\{\text{Open}\}$

9–12), which involves extending the sequence Λ of "test" user sets in line 10. Thus, the semantics of policy (6) is given by

$$\forall R^{\text{test}}_{1,1}, R^{\text{ref}}_{1,1}\subseteq Role_1;\ldots;R^{\text{test}}_{1,n}, R^{\text{ref}}_{1,n}\subseteq Role_n \bullet \big(filter_1(R^{\text{test}}_{1,1}, R^{\text{ref}}_{1,1},\ldots,R^{\text{test}}_{1,n}, R^{\text{ref}}_{1,n})\wedge [a^{\text{test}}_1, b^{\text{test}}_1]\bowtie_1[a^{\text{ref}}_1, b^{\text{ref}}_1]\big) \rightarrow \big(post_1(\text{RBAC}')\wedge\ldots\big),$$

where $[a^{\text{test}}_1, b^{\text{test}}_1], [a^{\text{ref}}_1, b^{\text{ref}}_1]$ are computed as shown in lines 5 and 6 of Algorithm 1, RBAC' is the state of the RBAC system after the reconfiguration from line 8, and the ellipses stand for the descriptions of rules 2, 3, …, N of the policy.

The intuition behind this application of $rule_i$ is that $R^{\text{test}}_{i,1}$, $R^{\text{test}}_{i,2}, \ldots, R^{\text{test}}_{i,n}$ include users whose behaviour with respect to the property encoded by ϕ_i is being examined by comparison to the value of the same property for the "reference" users from $R^{\text{ref}}_{i,1}$, $R^{\text{ref}}_{i,2}$, …, $R^{\text{ref}}_{i,n}$. The values of Φ_i for the test and reference users cannot be calculated precisely, so the observations associated with the available traces are used to obtain two confidence intervals for ϕ_i. If these confidence intervals do not overlap, and are in the relationship indicated by $\bowtie_i$, then the behaviour of the examined users is suspicious and the RBAC system is dynamically adapted to satisfy $post_i$. The $N>1$ rules of a policy (6) are applied in order, with $rule_{i+1}$, $1\leq i<N$, applied if and only if the application of $rule_i$ resulted in RBAC changes, so a *policy* (6) is applied by invoking the application of its first rule, as shown by function APPLYPOLICY from Algorithm 1. The rationale is to further examine the behaviour of suspicious users, e.g. by establishing additional properties for these users or by calculating higher confidence intervals for the same property, with further actions taken to restrict their access permissions if necessary.

Example 3. Table II shows how the informal policies from Table I can be formalised. For example, policy P1 contains a single rule whose test and reference traces are constrained by $filter_1$ to the traces of an individual client user ($R^{\text{test}}_{1,1}=\{u\}$) and the traces of the remaining client users ($R^{\text{ref}}_{1,1}=Client\setminus\{u\}$), respectively. The policy PCTL formula $\Phi_1=\mathcal{R}^{\text{reopened}}_{=?}[\text{F End}]$ indicates that confidence intervals for the expected number of Reopen operations performed on a (generic) ticket should be calculated, with confidence level $\alpha_1=0.95$. Finally, $\bowtie_i=>$ requires that RBAC adaptations are made if the 0.95-confidence interval for the test traces (i.e. for an individual client) is larger than 0.95-confidence interval for the reference traces (i.e. for the remaining clients); and the postcondition $perm'(u)=perm(u)\setminus\{\text{Open}\}$ requires that the offending client must lose the permission to perform Open activities after these RBAC adaptations.

D. RBAC adaptations

We selected the RBAC adaptation actions used to achieve the postconditions from saRBAC policies based on the functions defined by the RBAC specification [6]:

- creation and maintenance of elements and relations (e.g., add/delete of user/role, the assign/deassign of users and roles, and grant/revoke of permissions to roles);
- supporting user activities that are dynamic, i.e. that are performed as part of user sessions (e.g., create/delete session, add/drop active role, and check access); and
- reviewing the results of the previous actions, i.e., functions for making queries on the basic sets and relations (e.g., AssignedRoles, RolePermissions, SessionRoles, UserPermissions, UserOperationsOnObject).

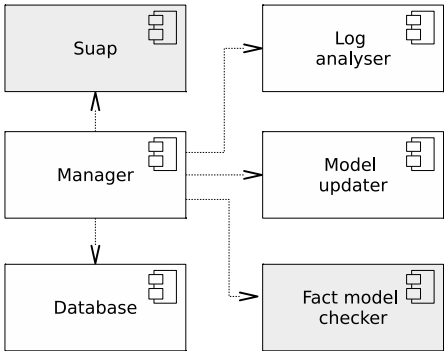

Fig. 3. saRBAC architecture (existing components are shaded)

Using these functions, saRBAC postconditions are achieved by Removing (i.e., disabling) elements from sets, and Adding (assigning) or Removing (revoking/deassigning) relations between users, roles, and permissions. Note that saRBAC disables instead of deleting RBAC elements as part of its adaptation actions, facilitating the reactivation of these elements by a system administrator upon reviewing the reasons for the adaptation. To deal with cases where a permission or permission set should be removed from a user, saRBAC employs restricted versions of the role without that particular permission (set). In our implementation, these restricted roles are identified by pre-analysing the adaptation policies, and are added to the system before the policies are actually set up.

Example 4. Consider again the adaptation policies from our Ticket System case study. The postcondition from the first (and only) rule of policy P1 requires the removal of the Open permission from offending client users. As such, our saRBAC approach creates a new role $Role_4 = ClientOpenRestricted$ during its pre-analysis stage. This new role is then used whenever policy P1 is applied to an offending user $u \in Client$ at runtime, by removing user u from the $Client$ role and adding him or her to the $ClientOpenRestricted$ role.

V. Implementation

We implemented a prototype for saRBAC in Python following a service-oriented architecture (Figure 3). This allowed the deployment of saRBAC components across several processing nodes, and thus the parallel analysis of multiple users. The prototype was integrated into the SUAP system as an advisory tool for IFRN management.

The *Manager* component is responsible for coordinating the analysis process according to Algorithm 1. This component interfaces with the system under analysis, represented by the *SUAP* component. The *Manager* is able to obtain logging information from *SUAP* containing the execution traces of its business processes, and information about its RBAC access control policy. Besides the information retrieved from *SUAP*, the *Manager* receives as input the parametric Markov model and policies definition. In its current implementation, the *Manager* component also receives as input the set of users to be analysed as part of the received policy definition.

Based on the log records obtained from *SUAP* the *Manager* builds the set of traces T for the business process being analysed. The *Log Analyser* component receives filter definitions as input, which are applied to the traces T. Filter definitions have been implemented using the keyword argument mechanism of the Python programming language. This component then produces the subsets of *test traces* T_i^{test} and *reference traces* T_i^{ref}. The *Model Updater* component analyses these traces in order to instantiate the parametric Markov model from Figure 2 to reflect the behaviour of observed users.

Finally, we use the FACT model checker [10] to compute confidence intervals for the property being evaluated. FACT uses the parametric probabilistic model checking functionality of PRISM to obtain a closed-form expression for the property, and the algorithms introduced in [9] to obtain the required confidence intervals based on the process traces. The *Manager* then follows the policy being evaluated, comparing the confidence intervals computed by FACT and yielding an RBAC adaptation recommendation if needed. When a policy contains more than one rule, the *Manager* processes one rule at a time. The *Database* component stores the data produced during the analysis process.

VI. Evaluation

To evaluate saRBAC, we conducted a series of experiments using our prototype implementation with real data from SUAP. The experiments aimed to show the feasibility of saRBAC to the IFRN IT management directorate. This section summarises the main results obtained in these experiments, divided in three parts. We begin by describing the experiments conducted to calibrate the confidence levels used for each policy (Section VI-A), followed by results about the use of the approach after calibration (Section VI-B). Finally, we present experiments that assess the performance of saRBAC (Section VI-C).

A. Preliminary Experiments and Confidence Level Calibration

For each policy it is necessary to calibrate the confidence level used to identify deviations in user behaviour whilst minimising the number of false positives. Automatic calibration of confidence levels remains a topic for further work and therefore a manual calibration was carried out as follows.

Data collected for 63 users of the system was considered over a time window of 30 days and, for users with the $Client$ role, three different subsets were considered. Initially, we selected users that are already known by the IT team, either in a positive or negative way. These are referred in the rest of the section as the subset of "famous" users. A second subset was defined as the most active users in which we selected the top users with the largest volume of handled tickets. Finally, we selected a subset of random users, comprising two client users. For the $Support$ role, besides the subsets of famous and most active users, we chose a subset of five users with a similar volume of tickets as the famous users, and a subset of random users, thus increasing the sample diversity. We also created several synthetic users to exercise specific policies deemed important by the IT managers, but which were not "triggered" by any current support user, e.g. policy P3.

TABLE III
RESULTS OBTAINED FOR THE POLICIES TARGETING THE *Client* ROLE

User type	User ID	P1 α=0.9	P6 α=0.9	User ID	P1 α=0.9	P6 α=0.9
famous	101701	0.8	0.95	101703	0	0
	101702	0.95	0.95	101704	0.9	0
random	101710	0.95	0	101711	0	0.8
most active	101705	0.85	0.95	101708	0	0
	101706	0.95	0.95	101709	0	0
	101707	0.8	0.95			

Tables III and IV present the results for these users in the *Client* and *Support* roles, respectively. Both tables contain an anonymised ID for each user, and the policies being considered with their confidence level thresholds, i.e. the confidence levels for which we deem that a user triggers a particular policy. For each user and policy, we computed confidence intervals for the examined behaviour characteristic at multiple confidence levels, and the tables report *the highest confidence level for which the policy is triggered*, i.e. the confidence level at which the results obtained from the test traces is outside the interval obtained for the reference traces. A '0' indicates that the user does not trigger the policy at any confidence level between 0.75 and 0.95. To decide the confidence level threshold of each policy, we started to compute confidence intervals at a confidence level $\alpha = 0.95$ and analysed the obtained results, identifying any false positive/negatives.[3] Next, we lowered the confidence level in steps of 0.05 and repeated the analysis. The results were discussed with the IFRN IT management, who were pleased with the effectiveness of the approach.

We start with the policies targetting the *Client* role, whose results are shown in Table III. Policy P1 represents clients who reopen tickets frequently, for which we defined as "famous" the users with a reopen rate above 50% of their opened tickets.[4] Considering these famous users, the results for users 101702 and 101704 were as expected, with both users triggering the policy (at 0.95 and 0.9 confidence levels, respectively). User 101703 was expected to not trigger the policy. Although user 101701 did not trigger the policy, it was expected to do so according to the initial manual analysis. Upon further analysis of this user, we confirmed that the result is correct, and the user indeed should not trigger policy P1; this discrepancy was due to an error in the (tedious) manual analysis. From the most active users, only user 101706 triggered policy P1. This was an unexpected result, that was confirmed by a close inspection of the user's activity traces. The same situation happened with user 101710 from the random subset. Policy P6 was designed to detect clients expensive to the Ticket Support System. The results from Table III are as expected for the famous users. For the most active users, the results were unexpected, but confirmed upon a closer look at the "offending" users (i.e. 101705, 101706, 101707).

The results for the policies targeting the *Support* role are presented in Table IV. Policy P2 handles support attendants with high rates of suspended tickets. Based on the number of tickets handled by a *Support* user, we defined as elevated a rate of suspension above 50% of the tickets handled. After an analysis of the results obtained, we selected a confidence level threshold of 0.8, which correctly identified users 201703 and 201708 as "offending" users.

[3]This involved an independent manual analysis of the activity of these users by the IT managers.

[4]Note that the rate of reopened tickets differs from the expected number of reopens per ticket, which is the metric used by policy P1, so not all "famous" users were expected to trigger the policy.

TABLE IV
RESULTS OBTAINED FOR THE POLICIES TARGETTING THE *Support* ROLE

User type	User ID	P2 α=0.8	P3 α=0.95	P4 α=0.85	P5 α=0.95	P5 α=0.99
famous	201701	0	0	0	0.95	0.95
	201702	0.75	0	0.8	0.99	0.99
	201703	0.85	0	0	0.99	0.99
	201704	0.75	0	0.8	0.99	0.99
	201705	0.75	0	0	0.99	0.99
most active	201706	0	0	0.85	0.99	0.99
	201707	0	0	0.85	0	0
	201708	0.8	0	0	0.99	0.99
	201705	0.75	0	0	0.99	0.99
	201703	0.85	0	0	0.99	0.99
similar	201711	0	0	0	0.99	0.99
	201712	0	0	0	0	0
	201713	0	0	0	0.99	0.99
	201714	0	0	0	0	0
	201715	0	0	0	0.85	0
	201716	0	0	0	0.99	0.99
synthetic	x1	0	0.95	0	0	0
	x2	0	0.95	0	0	0
	x3	0	0.9	0	0	0
	x4	0	0.85	0	0	0
random	201721	0	0	0	0.99	0.99
	201722	0	0	0	0	0

Policy P3 deals with *Support* users that cancel tickets without requiring more information. This is a very unusual behaviour for the Ticket Support System, and guarding against it was explicitly requested by the management team. This policy was not triggered by any of our users, and thus we created several synthetic users to analyse it. Four users were synthesized: two offenders, and two regular users with similar levels of activity. Since the results for this policy were obtained with synthetic users, it needs a careful revision, and possibly be run against the complete user base of SUAP.

Policy P4 deals with support users that open tickets on behalf of clients where the ticket is subsequently abandoned, another very suspicious behaviour for the Ticket Support System of SUAP. In our experiments, a confidence level of 0.85 proved adequate, being able to detect some unexpected cases with users 201706 and 201707. Upon further analysis of these particular users, we have confirmed that the results are indeed valid.

Finally, policy P5 addresses support users that are "lazy", which according to the management team is a support user with a high combined rate of suspended, canceled, reopened and reallocation tickets (cf. the "lazy" cost/reward structure from our DTMC model from Section IV-B). An analysis of the results obtained showed that a confidence level of 0.99 produced consistent results with all selected users, including those that were not expected to trigger such policy (as users

TABLE V
POLICIES FOR THE SUPPORT ROLE

User type	User ID	P2 α=0.8	P3 α=0.95	P4 α=0.85	P5 α=0.95	P5 α=0.99
famous	201701	–	–	–	–	–
	201702	#2	–	–	#2	#2
	201703	–	–	#1, #2	#1, #2	#1,#2
	201704	–	–	–	–	–
	201705	–	–	–	–	–
most active	201706	#2	–	–	#1, #2	#1, #2
	201707	–	–	–	–	–
	201708	#2	–	#1, #2	#1, #2	#1,#2
	201711	–	–	–	–	–
similar	201712	–	–	–	–	–
	201713	#2	–	–	#1, #2	#1,#2
	201714	–	–	–	–	–
	201715	#1	–	–	#1	–
random	201721	–	–	–	#1, #2	#1,#2
	201722	#1	–	–	#1	–

Key: #1=policy triggered on day 45; #2=policy triggered on day 60

201708 and 201721 for example).

We wanted saRBAC to be precise, robust to small variations in user behavioural patterns, and resistant to false alarms. The experiments presented so far assessed calibrated the confidence level threshold for each policy from Table II. Expectedly, reducing a threshold may increase the number of false positives, while lowering it may increase the number of false negatives. As an example, policy P1 presents false negatives when used with a confidence level of 0.95 (e.g. user 101704 from Table III). Accordingly, we chose a confidence level of 0.9 for the policy so as to achieve a correct result for all examined users. In contrast, for policy P2 we noted that a confidence level of 0.75 produced three false positives (cf. Table IV), but a confidence level of 0.8 was adequate.

B. Closed-loop Control Experiments

To investigate the effectiveness of saRBAC when run as a control loop, we carried out additional experiments using two datasets not utilised for calibration. These datasets comprised the log entries from days 31–45 and from days 31–60 of the 60-day period during which we monitored the system, respectively (recall that the calibration used the log entries from days 1–30). The new experiments focused on the *Support* role and assessed the robustness of the saRBAC calibration.

For each policy, Table V shows the users that triggered the policy for the analysis carried out using each dataset, i.e. on day 45 and on day 60. The results for policies P2 and P4 were consistent between the experiments for the two datasets, having correctly identified the expected "offending" users, and additionally several unexpected but still valid cases of offending support users such as 201706, 201713, 201715. For policy P5 with confidence level 0.99, the offending users 201715 and 201722 did not trigger the policy (false negatives). However, the two cases are correctly handled by the policy at 0.95 confidence level, so we firmed $\alpha = 0.95$ as the appropriate confidence level for this policy. For the reasons discussed in the previous section, we kept policy P3 although it again was not triggered by any user.

TABLE VI
RESULTS OBTAINED FOR THE PERFORMANCE EXPERIMENTS (REPORTED IN SECONDS, AS MEAN TIME $\pm$ STANDARD DEVIATION)

Policy	Log analyser	Model updater	FACT	Total
P1	0.00069 $\pm$0.00017	32.71 $\pm$0.53	30.65 $\pm$0.20	63.37 $\pm$0.68
P2	0.00059 $\pm$0.000010	32.46 $\pm$0.31	30.80 $\pm$0.21	63.27 $\pm$0.36
P3	0.00048 $\pm$0.0000087	32.45 $\pm$0.35	30.61 $\pm$0.1127	63.06 $\pm$0.3433
P4	0.00056 $\pm$0.00004	32.74 $\pm$0.50	30.69 $\pm$0.31	63.43 $\pm$0.81
P5	0.00060 $\pm$0.000024	32.52 $\pm$0.2201	30.71 $\pm$0.04907	63.24 $\pm$0.2875
P6	0.0005 $\pm$0.000007	31.86 $\pm$0.84	30.76 $\pm$0.20	62.63 $\pm$0.73

C. Performance Experiment

We also conducted experiments to assess the overheads of our approach. The experiments examined the performance of the saRBAC prototype when running on a single processing node, by measuring the execution times of its three main components, i.e. Log analyser, Model updater and FACT model checker. All experiments used real log data from SUAP over a 30-day time window (the same window considered for the calibration experiments), and involved a total of 919 tickets and 63 users (23 support users and 40 client users). All experiments were executed on 2.2 GHz Core i7 computer with 8GB of RAM, and were repeated three times.

Table VI presents the mean and standard deviation of the component execution times in seconds. The analysis of each policy takes approximately one second per user (above 60 seconds for the 63 users). While this overhead may seem too high for large business processes, it must be noted that saRBAC is meant to run infrequently (e.g. every few days) because the detection of insider threats within business processes with infrequent user activities requires multiple days of logs. Moreover, the performance of saRBAC can be improved dramatically by carrying out the essentially independent analyses required for different policies and different users concurrently, potentially after refactoring the architecture of our saRBAC prototype in line with existing self-adaptive system architectures [20], [21], [22], [23].

VII. RELATED WORK

The are several works on increasing the flexibility of access control mechanisms. These have been focused on adding dynamism to access control decisions, usually by means of incorporating risk and benefit values in the access control decision making process [24]. For example, Shaikh et al. [25] presented a method for risk-based access control decision, in which risk and trust are calculated based on historic user behaviour of granted access. Cheng et al. [26] employed Fuzzy Logic, which is quantified and used to define several thresholds according with risk tolerance. Based on trade-off analysis of risk versus benefit the solution grants access but with additional actions to mitigate risk depending on the threshold.

Examples of such additional actions include: stronger logging, extra charge for the user, different access levels token. Kandala et al. [27] presents an abstract formal model for expressing risk-based access control policies. The model is then applied to UCON access control model [28], which is extended to accommodate risk-awareness.

However, there is a distinction between dynamic access control decisions and dynamic modification of access control policies. The approaches based on the first consider that risk related information are encoded in the access control policy beforehand, restricting the decision making process to whether or not to grant access to a particular request. On the other hand, it is possible to notice a number of works moving towards the dynamic modification of the access control policies in response to detected situations.

Bijon et al. [29] presents a formalisation of an adaptive quantified risk-aware RBAC system, identifying how to utilize estimated risk values and thresholds in the access control decision making. Additionally, their approach also considers that risk values/threshold can be dynamically modified, identifying the need for monitoring, anomaly detection and risk re-estimation functions together with mechanisms to automatically revoke roles and permissions from users and roles respectively. Another approach is the Self-Adaptive Authorization Framework (SAAF) [7], which focus on adapting authorisation policies during run-time. SAAF has been demonstrated in the context of PERMIS authorisation system [30], in which access control policies are dynamically modified for dealing with insider threats, such as, an elevated number of downloads in a short time window. Ariadne [31], [32], [33] is an approach for dealing with security threats in Cyber-Physical systems by modelling the topology of the system (its physical objects and agents) together with its security requirements. This model is then used for conducting threat analysis and response planning at run-time about possible future changes in the topology, such as access of an agent to a particular area, for detecting violations of security requirements. Based on this, the approach suggests changes in the access controls rules for mitigating the identified threats.

Similar to those approaches, our work looks for means of adapting access control policies. However, different from them, we have focused on how to identify anomalous behaviour as trigger for adaptation. In this context, Legg et al. [1], [34], [35] presented an insider threat detection and analysis system. Their solution builds tree profiles of users based on different types of logging information, and applies a semi-supervised approach to assess how the current observations deviate from previously observed activities. In case of abnormality detection, the system raises an alarm. Probabilistic techniques have also been used for responding to network attacks, particularly as part of Intrusion Response Systems. [36] employs a Markov Decision Process as planning for deciding how to respond to a detected intrusion. Unlike this approach, our work focuses on insider threats operating inside trusted network boundaries, which the work in [30] does not consider.

Different from these approaches, we are looking at the application level, applying our approach to socio-technical systems based on business processes, where there is a strong human interaction. We also employ confidence level calculations for improving our decision making process. As we have more observations of normal behaviour, the confidence in our model ("profile") of normal behaviour and in making decisions based on it increases, allowing us to take different actions. Thus, adaptation is needed/useful. One clear observation from these related work is that access to real-world data is difficult, and thus, researchers synthesize data that are similar to that of a real-world enterprise, or use a subset of data points, or apply insider threat detection techniques to other problem domains. In our work, we gather a variety and volume of data observed in a modern real-world organization.

VIII. Conclusions

We introduced saRBAC, an approach to dynamically adapting the role-based access control system of a business process in order to mitigate insider threats. Our saRBAC approach uses a Markov model and execution traces of the business process to establish confidence intervals for key characteristics of user behaviour, and thus to identify users with harmful behaviour and to demote them to more restrictive roles. We implemented and evaluated saRBAC for a real IT support business process used by a large academic organisation. Because of the business criticality of this process, the preliminary evaluation results were obtained by running our implementation in an advisory operating mode, i.e. by providing suggestions of RBAC reconfigurations that IT managers from the organisation could examine.

As illustrated in Section VI, the preliminary evaluation results were positive, and as a next step of the project we plan to start using the system in fully self-adaptive operating mode for the policies with no or very low numbers of false positives, and with low numbers of false negatives. Additionally, we will examine further and fine-tune the policies that do not belong to this category, with a view to identify best practices for specifying saRBAC policies. In future work, we intend to apply saRBAC to other business processes (initially within the same organization) in order to assess and improve the flexibility of its policy specification formalism as well as its effectiveness at supporting a wider range of insider threat scenarios. Another area of future work is the development of a high-level language for the specification of saRBAC policies, to support users unfamiliar with the first-order logic formalism used by the current version of our approach.

References

[1] P. A. Legg, O. Buckley, M. Goldsmith, and S. Creese, "Caught in the act of an insider attack: detection and assessment of insider threat," in *2015 IEEE International Symposium on Technologies for Homeland Security (HST)*, 2015.

[2] E. Cole, "Insider threats and the need fot fast and directed responde," SANS Institute InfoSec Reading Room, Tech. Rep., 2015.

[3] A. M. George Silowash, Dawn Cappelli *et al.*, "Common sense guide to mitigating insider threats," CERT Carnegie Mellon, Tech. Rep., 2012.

[4] D. M. Cappelli, A. P. Moore, and R. F. Trzeciak, *The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crimes*, 1st ed. Addison-Wesley Professional, 2012.

[5] International Organization for Standardization, "ISO/IEC 27005:2011: Information technology – Security techniques – Information security risk management," 2011.

[6] ANSI, "Role based access control," NIST, Tech. Rep. ANSI INCITS 359-2004, 2004.

[7] C. Bailey, D. W. Chadwick, and R. de Lemos, "Self-adaptive federated authorization infrastructures," *Journal of Computer and System Sciences*, vol. 80, no. 5, pp. 935 – 952, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022000014000154

[8] M. B. Salem, S. Hershkop, and S. J. Stolfo, "A survey of insider attack detection research," in *Insider Attack and Cyber Security*. Springer, 2008, pp. 69–90.

[9] R. Calinescu, C. Ghezzi, K. Johnson, M. Pezzè, Y. Rafiq, and G. Tamburrelli, "Formal verification with confidence intervals to establish quality of service properties of software systems," *IEEE Trans. Reliability*, vol. 65, no. 1, pp. 107–125, 2016.

[10] R. Calinescu, K. Johnson, and C. Paterson, "FACT: A probabilistic model checker for formal verification with confidence intervals," in *TACAS*, 2016, pp. 540–546.

[11] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[12] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects of Computing*, vol. 6, no. 5, pp. 512–535, 1994.

[13] S. Andova, H. Hermanns, and J.-P. Katoen, "Discrete-time rewards model-checked," in *FORMATS 2003*, ser. LNCS. Springer, 2004, vol. 2791, pp. 88–104.

[14] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of Probabilistic Real-time Systems," in *CAV 2011*, ser. LNCS, vol. 6806. Springer Berlin Heidelberg, 2011, pp. 585–591.

[15] J.-P. Katoen, M. Khattri, and I. S. Zapreev, "A Markov reward model checker," in *Quantitative Evaluation of Systems*. IEEE Computer Society, 2005, pp. 243–244.

[16] R. Calinescu and S. Kikuchi, "Formal methods @ runtime," in *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, ser. LNCS. Springer, 2010, vol. 6662, pp. 122–135.

[17] R. Calinescu, K. Johnson, and Y. Rafiq, "Developing self-verifying service-based systems," in *ASE*, 2013, pp. 734–737.

[18] R. Calinescu, S. Kikuchi, and M. Kwiatkowska, "Formal methods for the development and verification of autonomic IT systems," in *Formal and Practical Aspects of Autonomic Computing and Networking: Specification, Development and Verification*, P. Cong-Vinh, Ed. IGI Global, 2011, pp. 1–37.

[19] R. Calinescu, Y. Rafiq, K. Johnson, and M. E. Bakir, "Adaptive model learning for continual verification of non-functional properties," in *ICPE'14*, 2014, pp. 87–98.

[20] D. Garlan, S.-W. Cheng, A. C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, Oct 2004.

[21] R. Calinescu, "Model-driven autonomic architecture," in *Fourth International Conference on Autonomic Computing (ICAC'07)*, June 2007, pp. 9–9.

[22] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *Future of Software Engineering, 2007. FOSE '07*, May 2007, pp. 259–268.

[23] R. Calinescu, "Implementation of a generic autonomic framework," in *Fourth International Conference on Autonomic and Autonomous Systems (ICAS'08)*, March 2008, pp. 124–129.

[24] R. W. McGraw, "Risk adaptable access control (RAdAC)," National Institute of Standards and Technology, McLean and Clifton, VA, United States, Tech. Rep., 2009.

[25] R. A. Shaikh, K. Adi, and L. Logrippo, "Dynamic risk-based decision methods for access control systems," *Computers & Security*, vol. 31, no. 4, pp. 447–464, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404812000399

[26] P. C. Cheng, P. Rohatgi, C. Keser, P. Karger, G. Wagner, and A. Reninger, "Fuzzy multi-level security: An experiment on quantified risk-adaptive access control," in *IEEE Symposium on Security and Privacy (SP '07)*, 2007, pp. 222–230.

[27] S. Kandala, R. Sandhu, and V. Bhamidipati, "An attribute based framework for risk-adaptive access control models," in *Sixth International Conference on Availability, Reliability and Security (ARES)*, 2011, pp. 236–241.

[28] J. Park and R. Sandhu, "The uconabc usage control model," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 1, pp. 128–174, feb 2004.

[29] K. Z. Bijon, R. Krishnan, and R. Sandhu, "A framework for risk-aware role based access control," in *Proceedings 6th IEEE-CNS Symposium on Security Analytics and Automation (SAFECONFIG)*, 2013, pp. 462–469.

[30] D. W. Chadwick *et al.*, "PERMIS: A Modular Authorization Infrastructure," *Concurr. Comput. : Pract. Exper.*, vol. 20, no. 11, pp. 1341–1357, Aug. 2008. [Online]. Available: http://dx.doi.org/10.1002/cpe.v20:11

[31] C. Tsigkanos, L. Pasquale, C. Menghi, C. Ghezzi, and B. Nuseibeh, "Engineering topology aware adaptive security: Preventing requirements violations at runtime," in *IEEE 22nd International Requirements Engineering Conference (RE 2014)*, Aug 2014.

[32] C. Tsigkanos, L. Pasquale, C. Ghezzi, and B. Nuseibeh, "Ariadne: Topology aware adaptive security for cyber-physical systems," in *IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE 2015)*, vol. 2, May 2015, pp. 729–732.

[33] L. Pasquale, C. Ghezzi, C. Menghi, C. Tsigkanos, and B. Nuseibeh, "Topology aware adaptive security," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2014)*. ACM, 2014.

[34] I. Agrafiotis, J. R. Nurse, O. Buckley, P. Legg, S. Creese, and M. Goldsmith, "Identifying attack patterns for insider threat detection," *Computer Fraud & Security*, vol. 2015, no. 7, pp. 9 – 17, 2015.

[35] P. A. Legg, O. Buckley, M. Goldsmith, and S. Creese, "Automated insider threat detection system using user and role-based profile assessment," *IEEE Systems Journal*, no. 99, pp. 1–10, 2015.

[36] S. Iannucci and S. Abdelwahed, "A probabilistic approach to autonomic security management," in *2016 IEEE International Conference on Autonomic Computing (ICAC)*, 2016.