Advanced Software Engineering Project

Luca Brunetti 1967993
Niccolò Decembrini 2083867

June 21, 2025

Contents

1 Backend introduction **3**

2 WebRatio Project Design **3**

3 Admin's View **4**
- 3.1 Users area . . . 4
- 3.2 Settings area . . . 5
- 3.3 Sensors Area . . . 7

4 IDS operator **9**
- 4.1 MyAccount . . . 9
- 4.2 Notifications Area . . . 11
- 4.3 Events Area . . . 12
- 4.4 Traffic Area . . . 13
  - 4.4.1 Anomalous Traffic Details page . . . 14
  - 4.4.2 Legit Traffic Details page . . . 15
  - 4.4.3 Assign Event page . . . 16

5 Alert System Operator **18**
- 5.1 Anomalous Traffic Area . . . 18
- 5.2 Event Management Area . . . 20
- 5.3 MyAccount . . . 21

6 Conclusions **22**

Our final course project is a management system for a *Network-Intrusion Detection System* built with **WebRatio**. The system uses several inline sensors to capture, analyze, and store network traffic data in a database. The management interface is then used by various actors to access the data and perform different tasks.

1 Backend introduction

The backend consists of a Python script and a PostgreSQL database that constitute the IDS's data storage. The script implements a machine learning algorithm that simulates the behavior of an inline sensor: it retrieves a network record from the NSL-KDD99 Dataset, classifies it, and stores it in the database every 30sec.

2 WebRatio Project Design

The WebRatio project is organized into four distinct views, one for each system actor, and a login page.

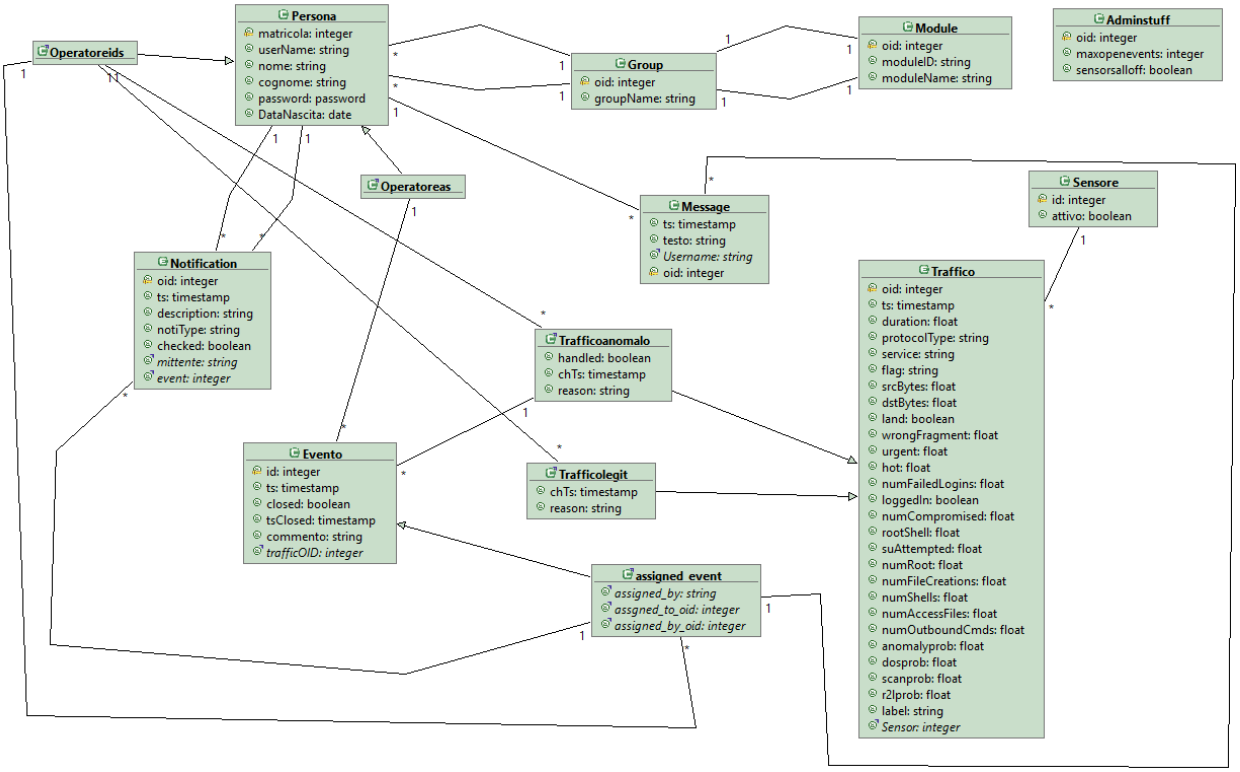


Figure 1: WR Class Diagram

In this context, an "event" represents the status of a traffic record that has been processed.

3 Admin's View

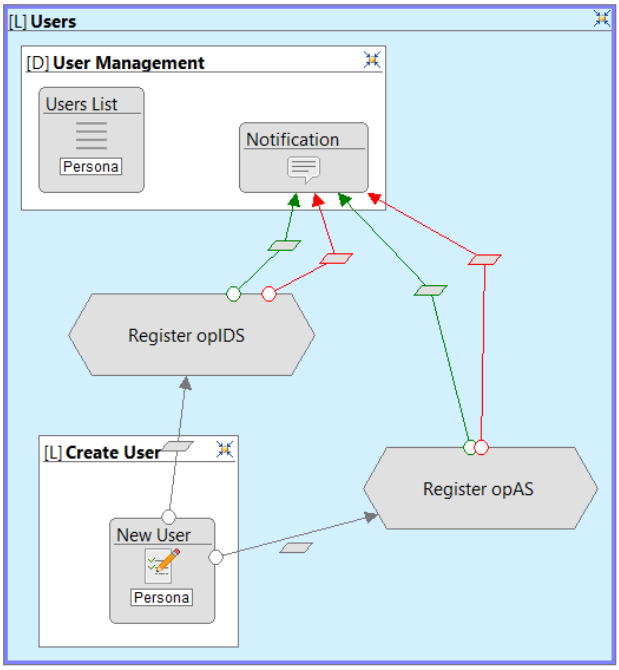
The Admin in our domain has several tasks, they have to register the operators, turn OFF/ON sensors, view/send notifications and set some very important things about the system.

Figure 2: Admin Use Case Diagram

The Admin's view got six landmarks. A ***Groups*** area is used to grant operators access to their right view, next we got a ***Logout*** and a ***Notifications*** area that will be seen in details in the IDSoperators view here. Lets see the others:

3.1 Users area

The administrator can view a list of all users along with their assigned groups (the user–operator derivation is a complete disjoint, so each user belongs to exactly one group). Admin can also create new accounts and designate the new user as either an IDS operator or an AlertSystem operator. We chose to grant this registration privilege exclusively to the administrator, rather than allowing users to self register, because in a real-world company scenario it makes little sense for employees to assign themselves roles.



Both the operations takes as input the user information via form and create the istance, the output prints an error/operation success message.

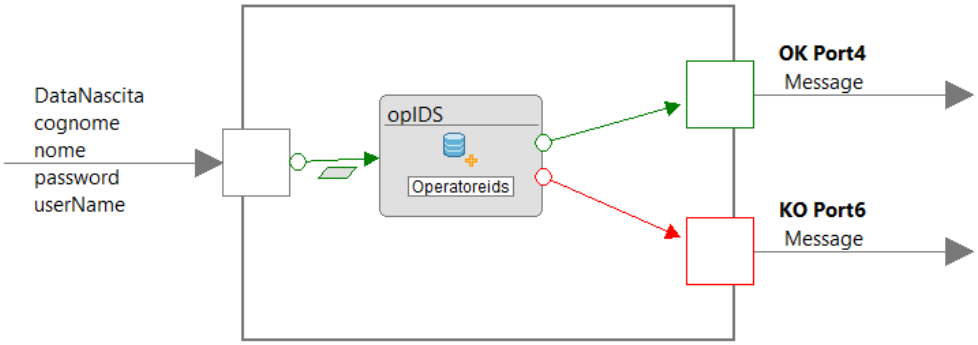
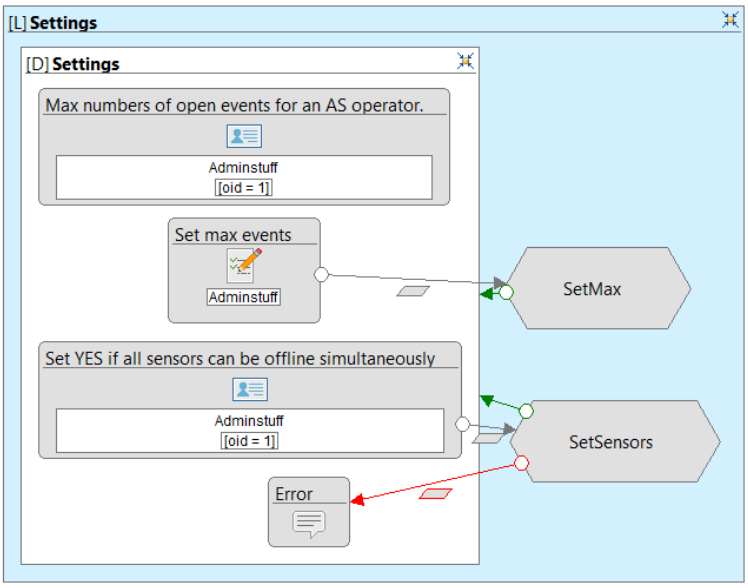


Figure 3: The *Register opIDS* operation. The opAS is analogous, differing only in the assigned group number.

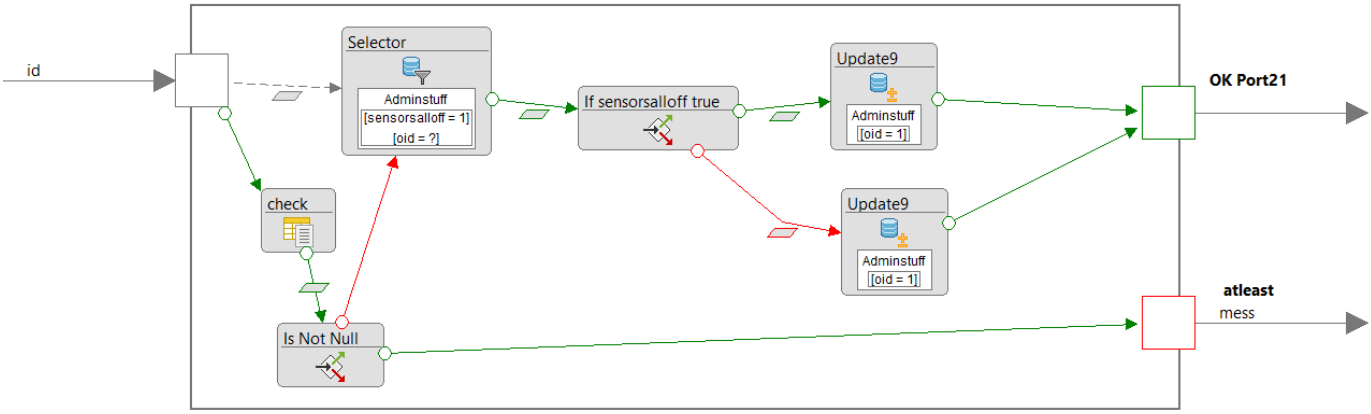
3.2 Settings area

In our own setup, we've got some subjective rules such as: *can all the network-scanning sensors go offline at once?* From a straight-up security point of view, you'd say "no," but during maintenance, the answer could easily be "yes." So it makes sense to let the admin decide when all sensors can be offline (remember that the IDS operators have access to sensors state). Another one can be– *How much open events can an AlertSystem operator handle?* This directly affects traffic-mitigation assignments. We decided to let the admin pick the maximum number of open events an AS operator can have—or even choose no limit; notice that **this setting does not represent a trigger or a full instances constraint!**

It's simply an assignment check to decide if the operator can take on another anomalous record. It doesn't retroactively invalidate existing assignments if you change the limit while someone's already over it.



So in this page the Detail components prints the current settings, the *SetMax* operations takes via form the number of open events and update the rule (leave blank and commit for no limit). The *SetSensors* operation is more tricky because there's an ambiguity: what if the *AllSensorsCanBeOff* rule is *True* with all the sensors off and we want to turn it to *False*?



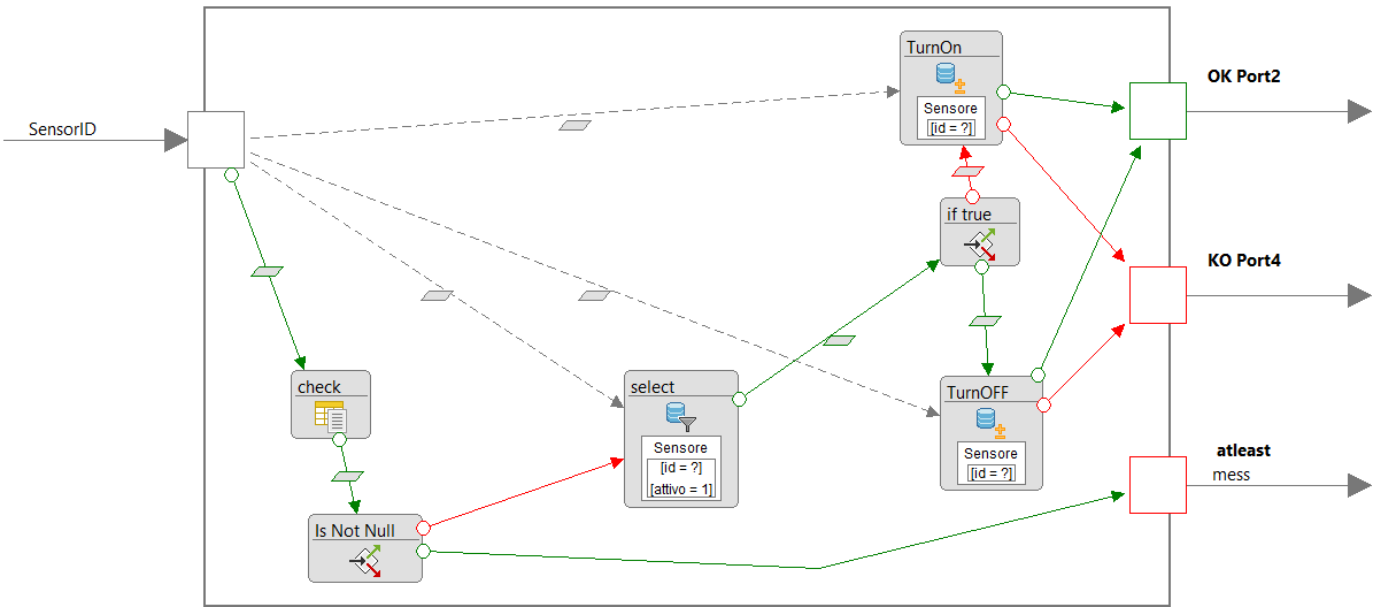
We have to make sure that the operation commit if there's at least one ON sensor when we wants to change the rule, the first two components implements this check, with the query as:

```
SELECT
  CASE
    WHEN NOT EXISTS (
            SELECT 1
            FROM   public.sensore
            WHERE  attivo = TRUE)
    THEN TRUE
    ELSE NULL
  END
```

Next the selector and the if components are needed to modify properly the rule (If False modify in True and viceversa).

3.3 Sensors Area

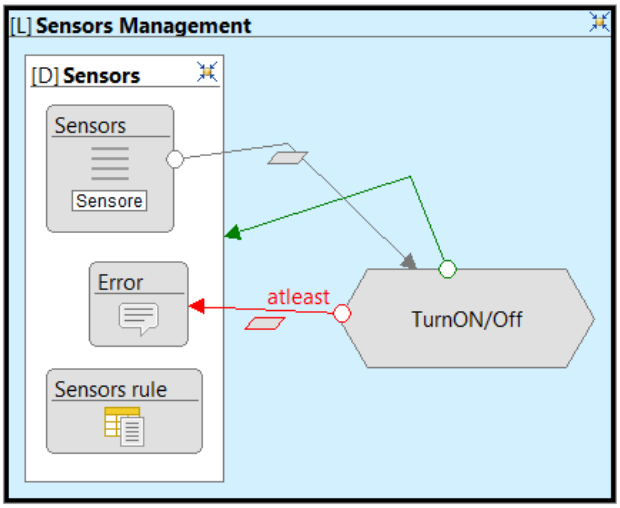
Here we can view the list of sensors with their current state and a query that prints the current status of the *AllSensorsCanBeOff* rule. The main part is the *TurnOn/Off* operation that acts as a switch on the current sensor status.



The operation takes as input the ID of the sensor we called the operation on and gives an error or commit as output. The first two components check if we are trying to turn off the only ON sensor while the *AllSensorsCanBeOff* rule is *False*, if this is true then we generate an error, with the query as:

```
SELECT
  CASE
    WHEN s.attivo = TRUE
     AND NOT EXISTS (
           SELECT 1
           FROM   public.sensore x
           WHERE  x.id <> :sensorID
             AND  x.attivo = TRUE)
     AND a.sensorsAllOff = FALSE
    THEN TRUE ELSE NULL
  END
FROM   public.sensore  AS s
AND  public.AdminStuff AS a
WHERE  s.id = :sensorID AND a.oid = 1
```

If we proceed with no errors the next components change properly the sensor status (If OFF set to ON and viceversa).



4 IDS operator

IDS operators are responsible for coordinating network security. They interact with network traffic by reanalyzing it to detect false negatives and false positives, and they assign operators to events whenever they determine that a threat requires rapid mitigation. Once an event has been assigned, they use an integrated chat feature to manage and coordinate the mitigation effort. They can also interact directly with the sensors.

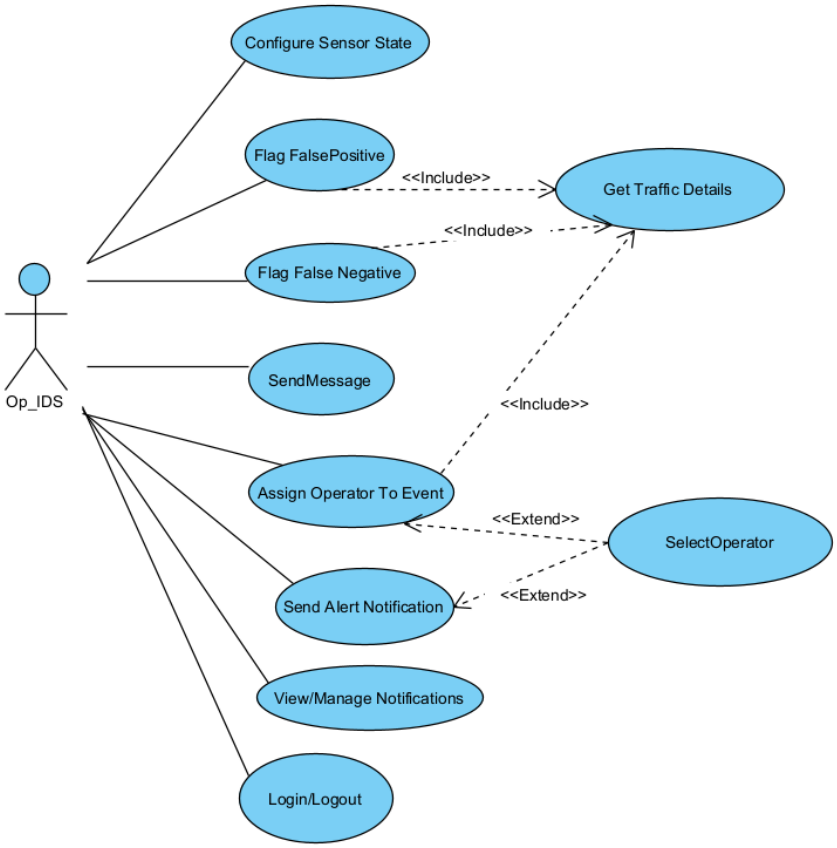
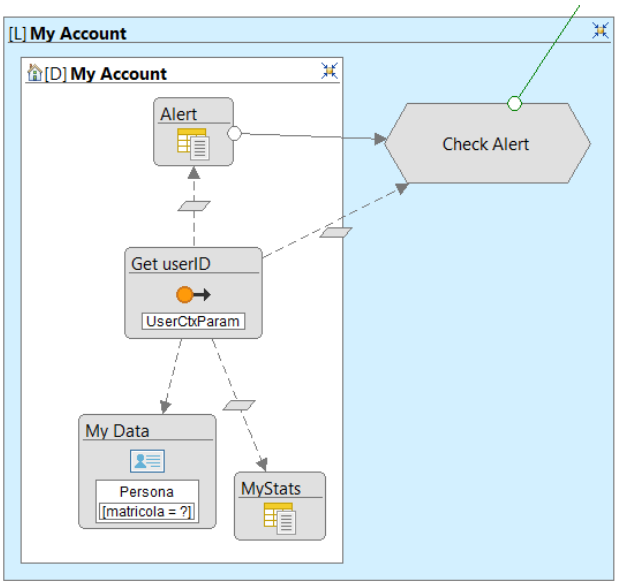


Figure 4: IDS operator Use Case Diagram

Lets see now the IDSop's View, it is composed by six landmarks:

4.1 MyAccount

This is the landing page shown immediately after login. It displays general user information—username, first name, last name, and date of birth—and provides an overview of events you've closed, events still open, and events assigned to you. It also alerts you to the number of unread notifications.



So the **Alert** query just print the number of the notifications directed to the user with the *checked* flag *False*.

```
SELECT CONCAT('You have ', COUNT(*), ' unread notifications')
FROM   public.notification AS n
WHERE  n.to_pp = :userID AND n.checked = FALSE;
```

The **MyStats** query just print the various assigned_events assigned by the user, the one with the *closed* flag *True* and *False*...

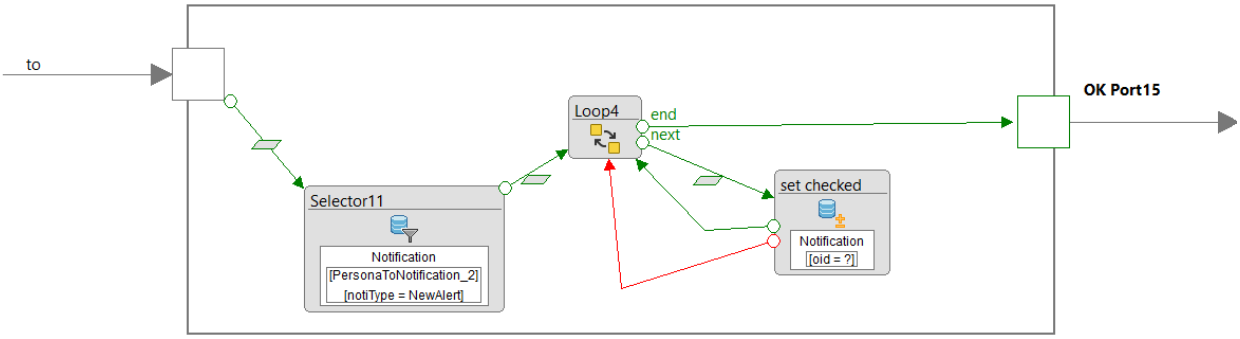
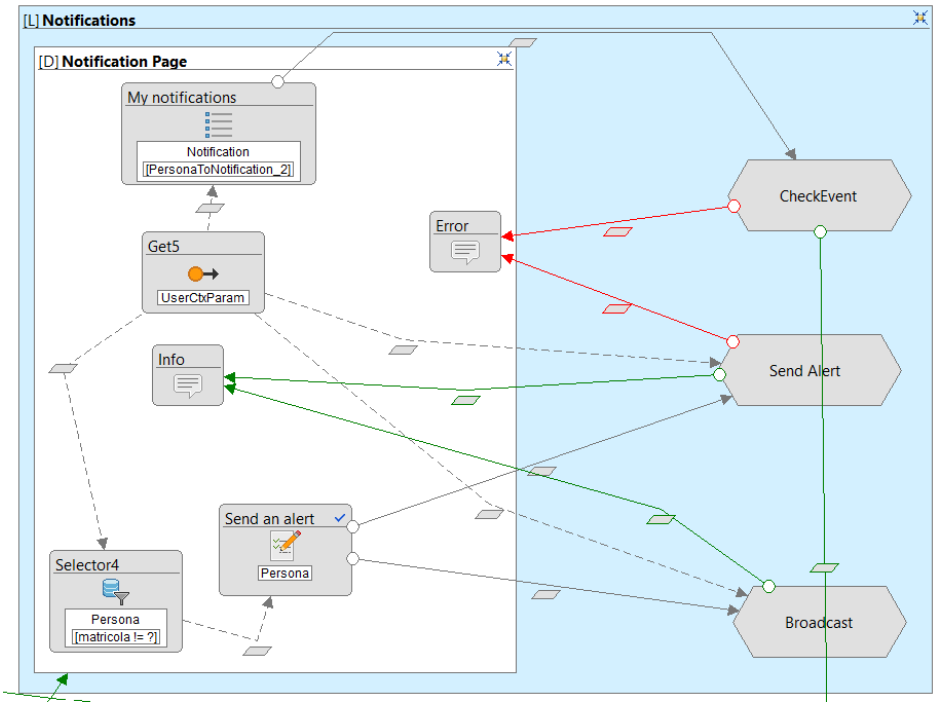


Figure 5: CheckAlert Operation

The *CheckAlert* operation takes in input the user ID and bring him his personal ***Notification*** page; the part inside basically takes the Array of notification IDs directed to the user whose type is "NewAlert" (notifications not related to any event, sent by another user) directed to the specified user and marks them as read.

4.2 Notifications Area



The List shows the User's notifications showing – timestamp, description (when the type is "NewAlert" the dscr is the alert-message itself, when the noti is event-related the description tells the user that something new is happening with one of his event [see the example below]), the sender and if the notification is read –

5/16/25 5:07:52 PM	Security policy updated – please review.	lucas01as	yes	check
5/16/25 5:07:52 PM	System maintenance scheduled at 22:00.	lucas01as	yes	check
5/13/25 7:14:29 PM	An event has been closed!	lucas01as	yes	check

Figure 6: The first two istances are "NewAlert" type, the third is event-related.

The first operation allows the user to view the event that generated the notification and marks it as read. Upon execution, it redirects the user to the respective ***AssignedEventPage***, or displays an error if the notification is not event-related. The remaining operation facilitates sending an alert:

- *Send Alert* takes as input the current user's ID and, via a form, the alert description and the recipient's user ID (selectable from a dropdown listing all operator usernames except the current user). Upon submission, it either displays an error message or confirms that the alert was sent successfully. For this operation selecting an operator is mandatory.
- *Broadcast* takes as input the current user's ID and the description via form, then loops over all operator user IDs—excluding the current user—and sends the alert to each of them.

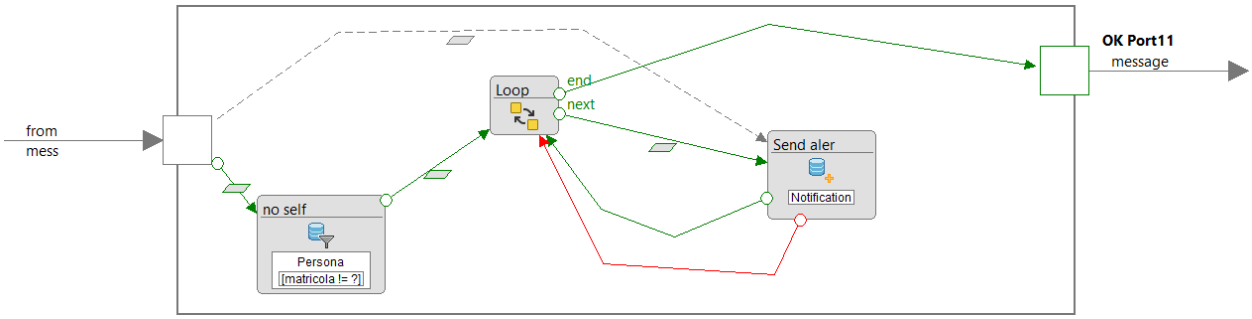
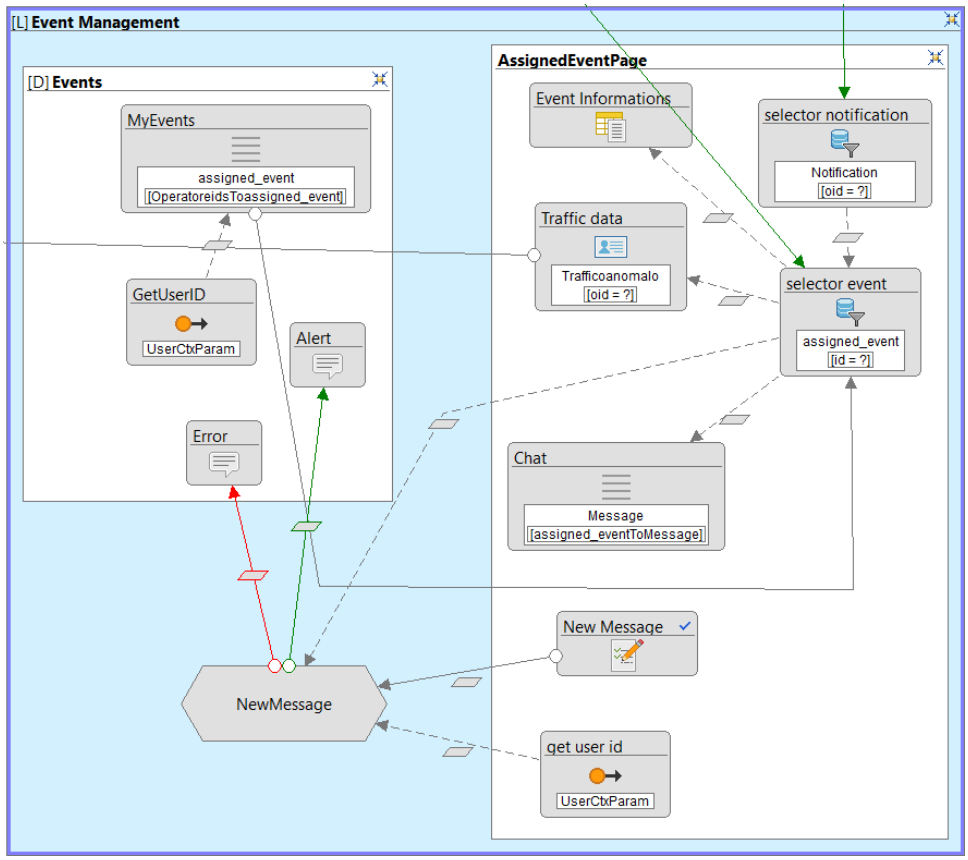


Figure 7: Broadcast operation – The selector give us the array of user IDs different from the current user's one.

4.3 Events Area



The ***Events*** page displays the list of events assigned by the user, clicking an event opens its corresponding ***AssignedEventPage***. The first query retrieves information such as the user who assigned the event, the assignment timestamp, and—if the event is closed—the reason for closure and the closing timestamp. Next, the Detail component displays key features of the anomalous record, followed by the chat between the operators handling the mitigation. The *NewMessage* operation allows the user to send a message by entering it into a form.

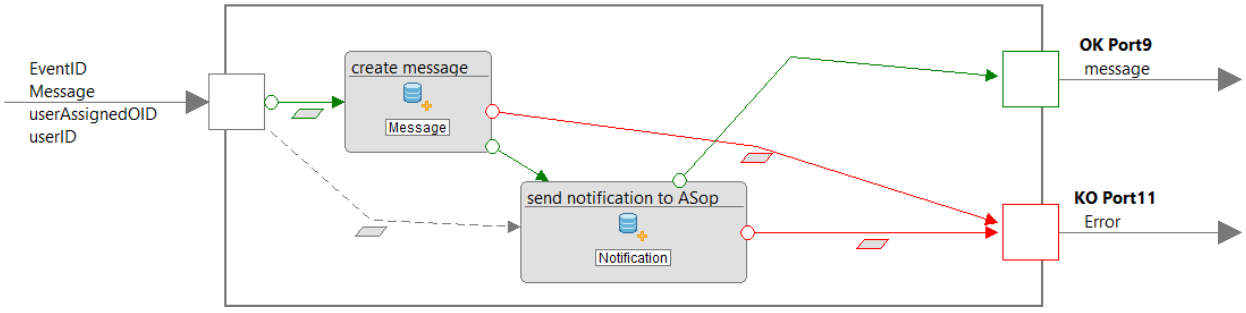
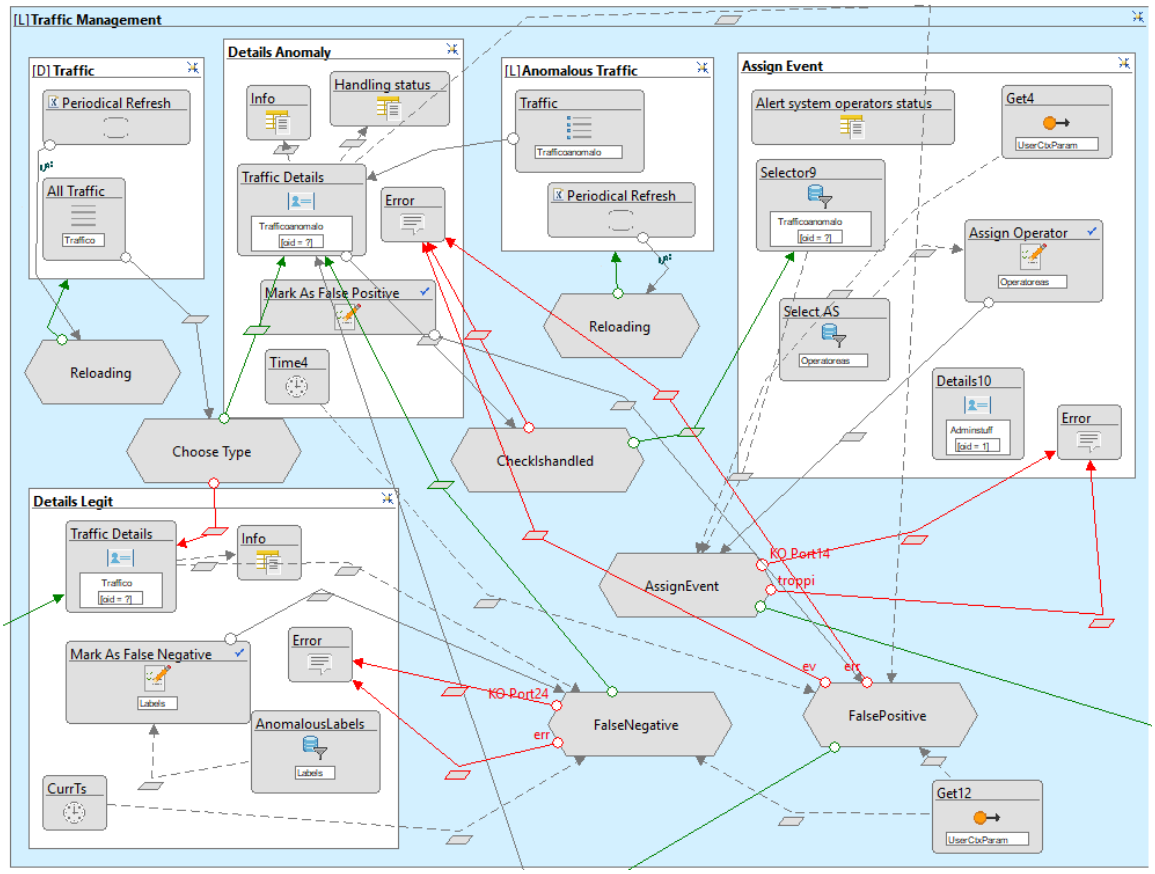


Figure 8: The *NewMessage* operation. The second component sends a "NewMessage" notification to the user receiving the message.

4.4 Traffic Area

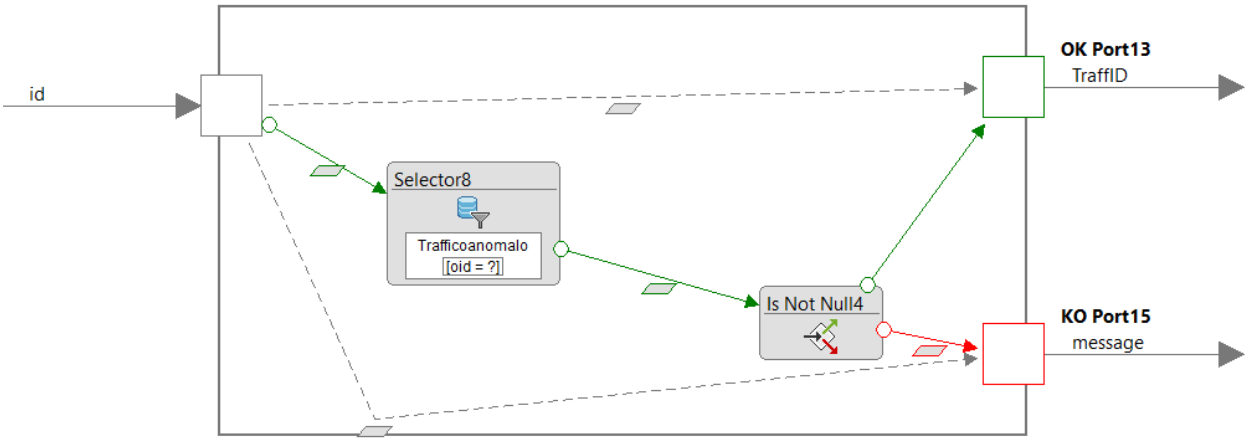


The default page of the area is the ***Traffic*** page. It presents a list of traffic records showing only the essential information: OID, timestamp, protocol type, service, classification label, and the sensor that captured each record. As mentioned earlier, each record is "captured" every 30 seconds by the machine learning algorithm simulating real sensors; the custom *Periodical Refresh* component automatically refreshes the list, allowing users to view newly captured records in real time. It is made with the custom unit template below, with a default refresh window of 30sec:

```
<wr:LayoutParameter name="interval" label="Interval" type="integer" default="30000">
 The timeout expressed in milliseconds
</wr:LayoutParameter>
[% setHTMLOutput() %]
[% def periodicalLink = unit.selectSingleNode("layout:Link") %]

<script>
 if(window["<wr:Id context="unit"/>"]){
clearInterval(window["<wr:Id context="unit"/>"]);
}
window["<wr:Id context="unit"/>"] = setInterval(function(){[% printRaw(getAjaxRequestCall(periodicalLink, "index")) %]}, [%= params["interval"] %] );
</script>
```

The *Reloading* operation is just the input port connected to an OK port bringing back the user to the page.
The ***Anomalous Traffic*** page is similar, but displays only records labeled as attacks. Each record also includes a boolean flag indicating whether it is currently being handled or has been closed. After clicking on a record, you're redirected to its corresponding ***Details*** page. Note that the Details pages for legitimate and anomalous records differ: each displays different attributes and offers different operations. Therefore, when navigating from the general ***Traffic*** page, the *Choose Type* operation determines and loads the appropriate ***Details*** page for that record:



4.4.1 Anomalous Traffic Details page

In this page, the ***Details*** component shows all the information related to the traffic record, such as the feature values, the probabilities for each label produced by the ML algorithm, and

the sensor that captured it. The query component indicates whether this record has already been handled. The first operation, *CheckIsHandled*, allows the IDS operator to assign a new event to an ASop: it takes the anomalous traffic record OID as input and either returns an error if the record is already linked to an event or redirects the user to the ***Assign Event*** page.

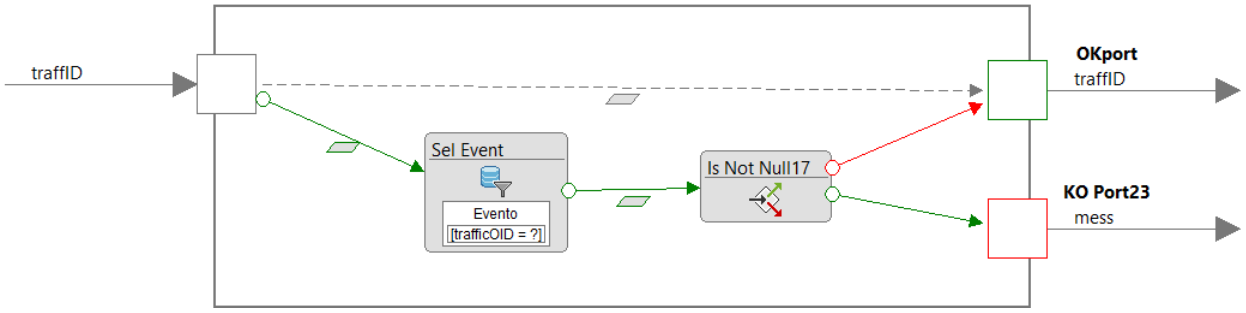


Figure 9: *CheckIsHandled* operation.

The last operation, *FalsePositive*, allow the IDSoperator to report the current record as a false positive. The operation takes as input the current timestamp and the reason why this traffic is safe (submitted by the operator).

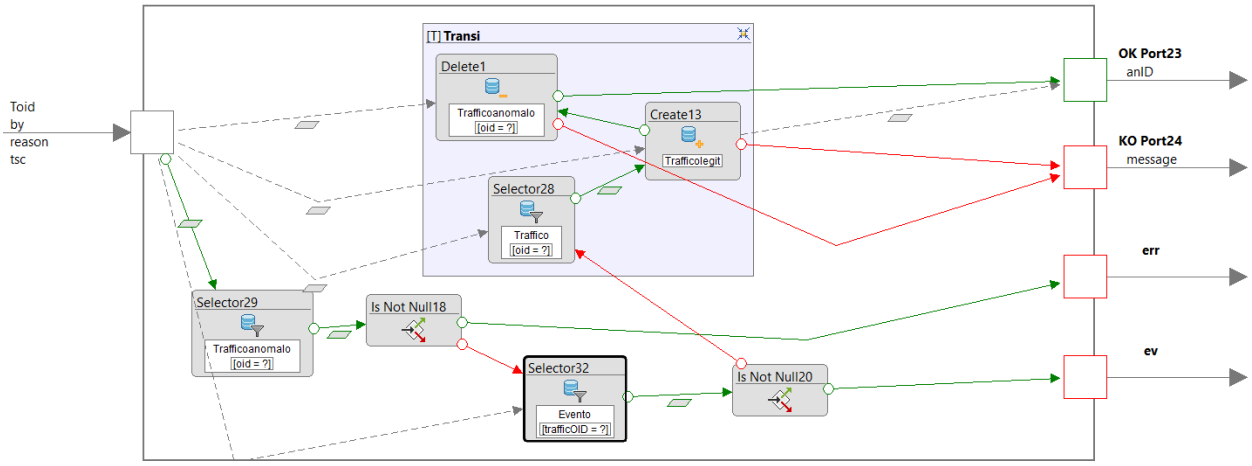


Figure 10: *FalsePositive* operation.

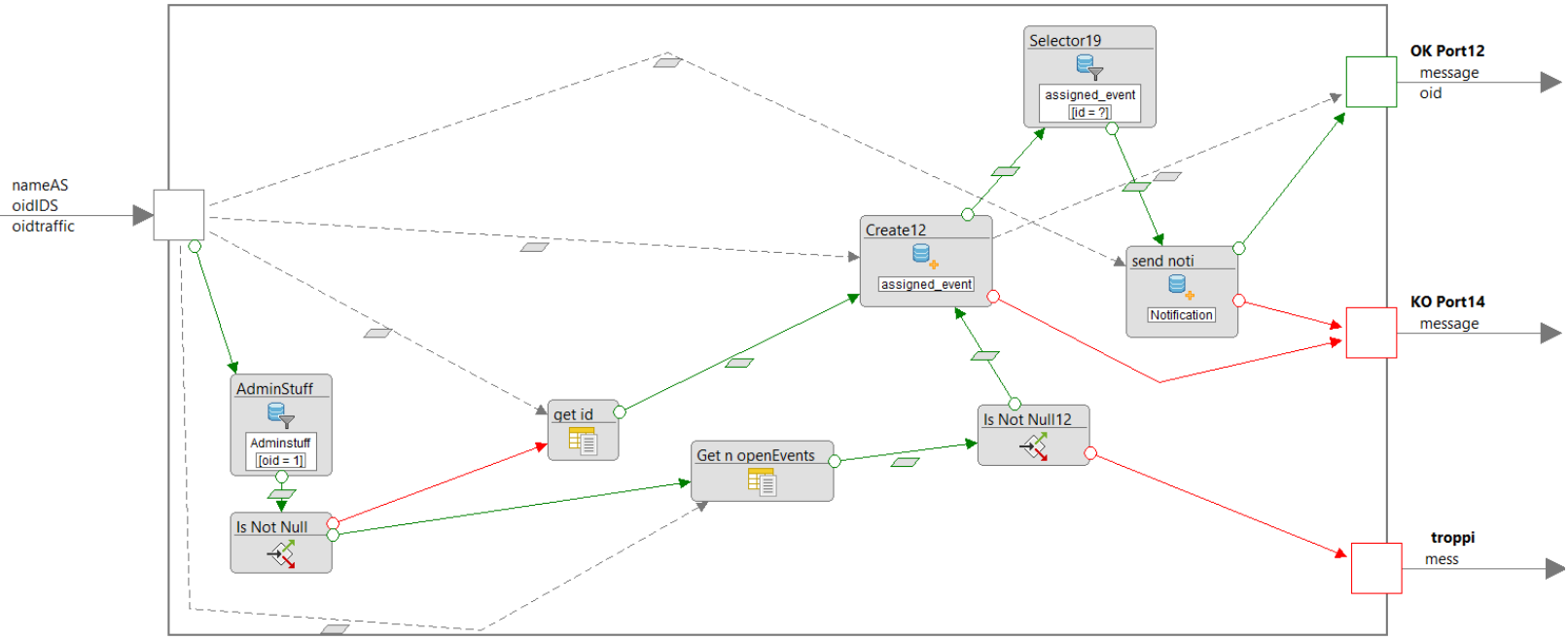
Here before committing any action there re two checks: the first one generates an error if the current record has already been reported as a False Negative, the second one checks if the traffic is/was involved in an event. If both the checks passes we proceeds switching the subclass of the record to TrafficLegit with *"Normal"* as label.

4.4.2 Legit Traffic Details page

This page has the same structure of the previous one, with the operation *FalseNegative* that takes in input the anomalous label aswell.

4.4.3 Assign Event page

This page allows an IDS operator to assign an AlertSystem operator to mitigate an anomalous record. First, the form presents a selection field listing all AS operators. A query then displays, for each operator, the number of open events they are currently handling. Finally, the Details component shows the current *MaxOpenEvents* rule. The *AssignEvent* operation takes three inputs—the anomalous record OID, the IDS operator OID, and the chosen AS operator username—and either returns an error if the AS operator cannot handle another event or redirects to the new Event page.



So the first two components check if the rule is set to NULL, so the number of open events has no limit, if so the operation proceeds to creating the event (the get id query takes the ASop's username as input and return the ID, for unknown reasons this operation doesnt work if we pass as input directly the OID from the selection field...). If the rule is not set to NULL the next query actually check if the ASop has more open events then is permitted, the SQL is:

```
SELECT
  p.matricola,
  p.username
FROM   public.persona p
LEFT   JOIN public.evento e
  ON   e.operatore = p.matricola
  AND  e.closed    = FALSE
WHERE  p.username = :userID
GROUP  BY p.matricola, p.username
HAVING COUNT(e.id) < (
```

```
11      SELECT a.maxOpenEvents
12      FROM   public.AdminStuff AS a
13      WHERE  a.oid = 1)
```

If the operator has more than permitted, the "Is Not Null" will generate an error, either we proceed creating the event and send a "NewEvent" notification to the assigned operator.

The IDSop's View ends with the ***Logout*** area and the ***Sensor Management*** area (same as the one seen in the Admin View here).

5 Alert System Operator

The **AlertSystem operator** plays a crucial role in network security, actively mitigating ongoing attacks. In our system, an AS operator engages with anomalous traffic either by being assigned to events or by volunteering to handle them. They communicate with the assigning operator via the integrated chat and close events by submitting a report once mitigation is complete.

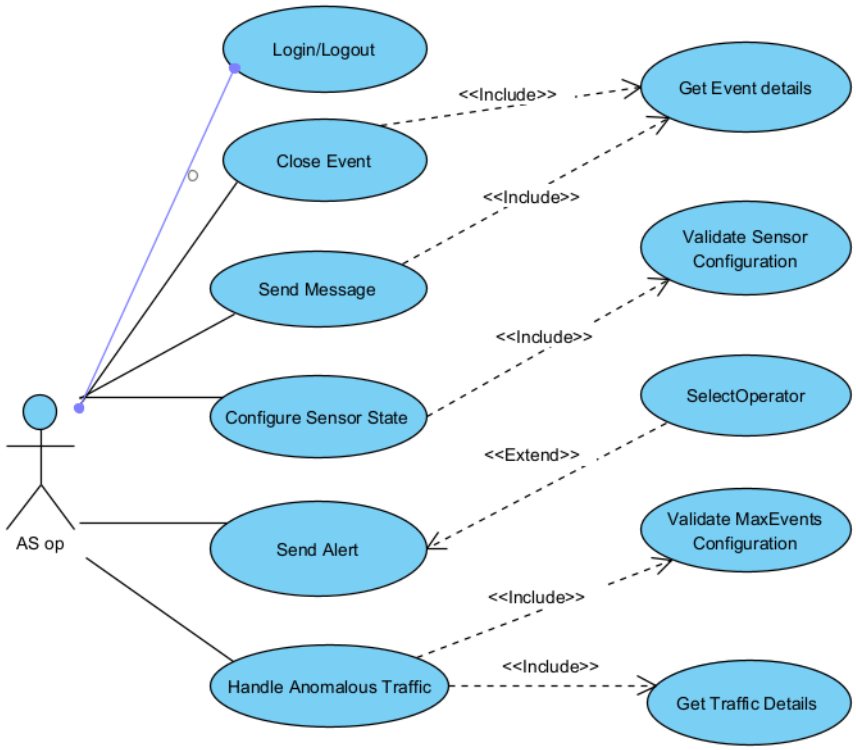
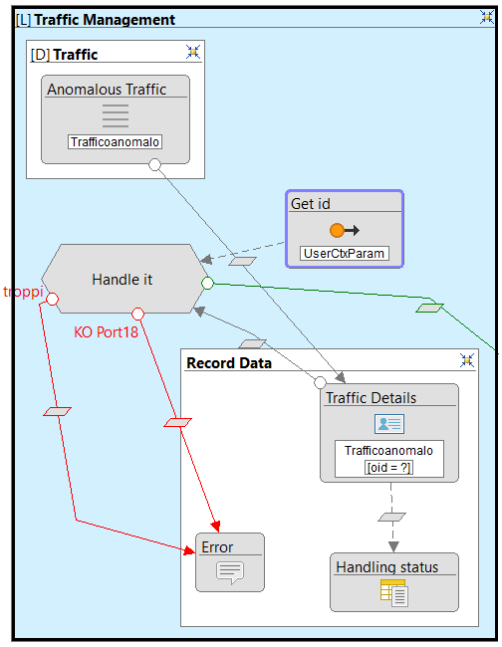


Figure 11: Alert System operator Use Case Diagram

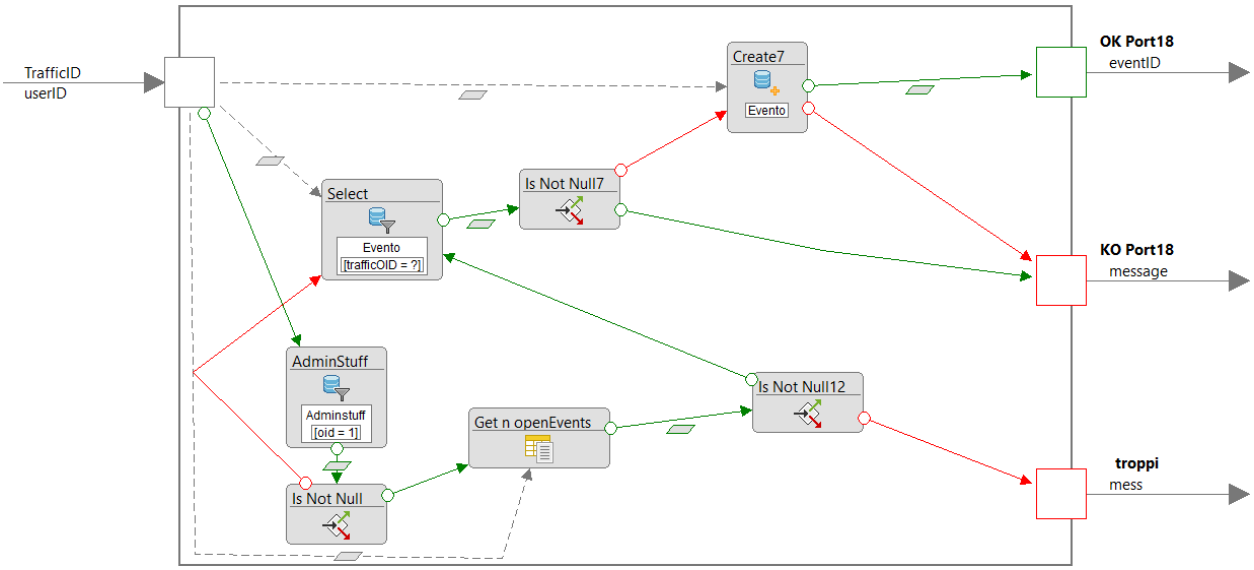
His view is composed by five landmarks. The ***Logout*** and the ***Notifications*** area seen before here. Lets see the others:

5.1 Anomalous Traffic Area

The first page grants access to the list of anomalous records, displaying each record's OID, timestamp, protocol type, service, classification label, and a boolean flag indicating whether it's already linked to an event. Clicking a record opens its corresponding details page.

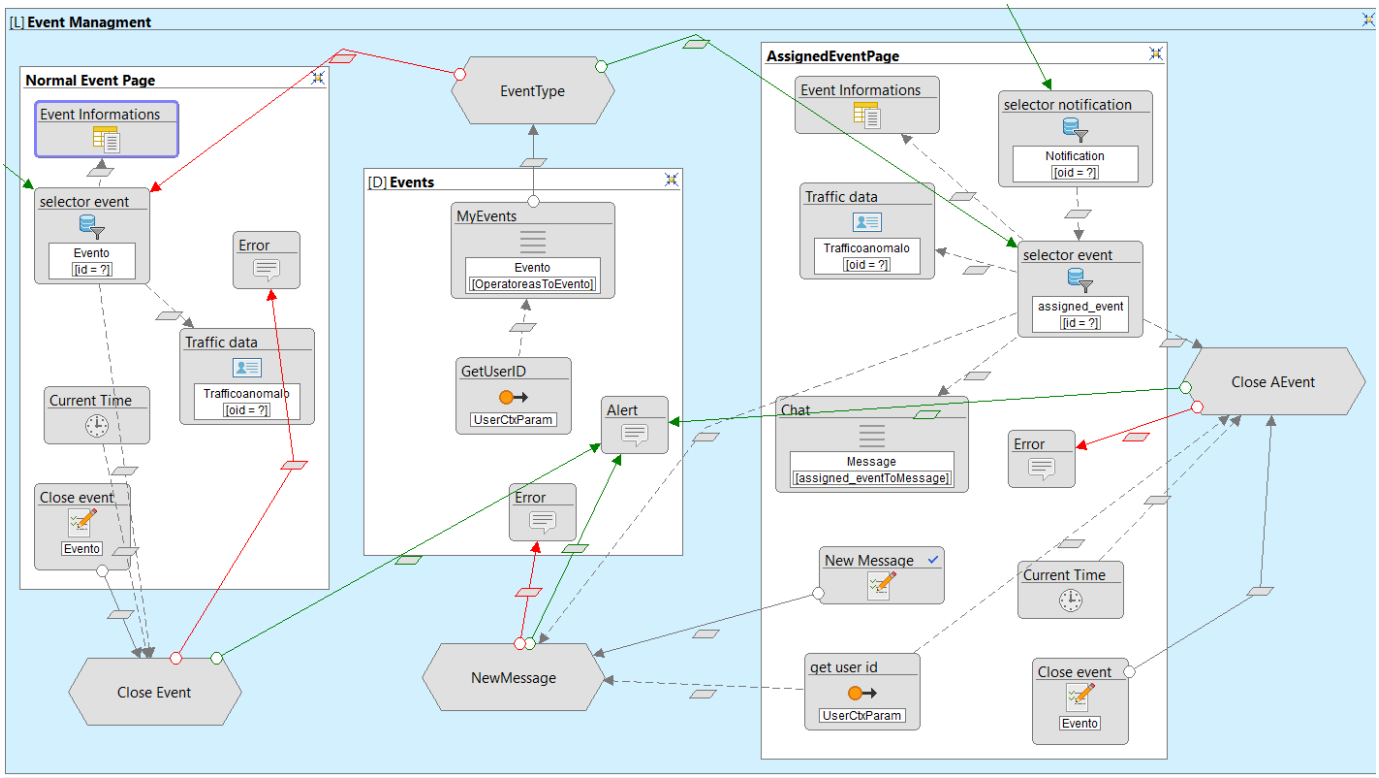


The details page shows again if there's already an event for this record via query. The *HandleIt* operation takes the current user's OID and the traffic record's OID as input; it either returns an error—if the operator exceeds the *MaxEvents* limit—or commits the operation and redirects the user to the new event page.



The operation is similar to the one seen here where the first three components determine if the user is exceeding the limit, if true raise error else commit. Notice that the second "Is Not Null" generates an error if there is already an event for the traffic record.

5.2 Event Management Area



The ***Events*** page displays the list of events associated with the user. Clicking an event opens its corresponding Details page. Events assigned to the operator differ from those handled directly—first of all, assigned events include the chat with the IDS operator who created them. The *EventType* operation takes an event's OID as input and redirects the user to the appropriate Details page.

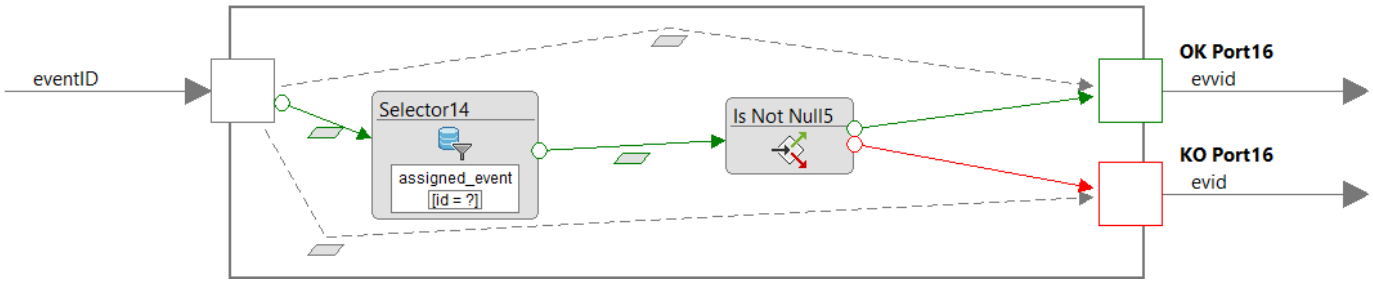


Figure 12: *Event Type* operation.

Both pages display event details such as the creation timestamp and, if assigned, the IDS operator who created the event, along with the anomalous record information. The *NewMessage* operation works as described here. The *CloseEvent* and *CloseAEvent* operations both take the current timestamp, the event's OID, and a closure comment (via form) as input;

they return an error if the event is already closed, or commit the closure otherwise. The *CloseAEvent* operation additionally sends an "Assigned Event Closed" notification to the involved IDS operator upon successful closure.

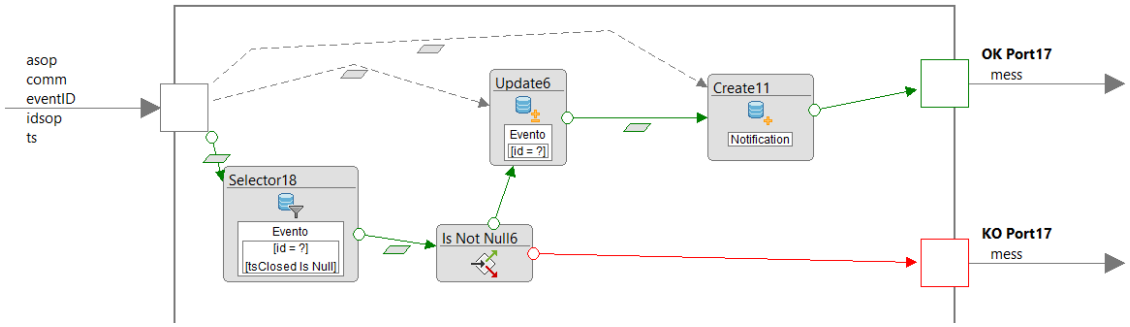


Figure 13: *Close AEvent* operation.

5.3 MyAccount

Is the same as the IDS operator home page. Differs only in the query, showing some statistics about the user work:

```
ELECT
  COUNT(*)                          AS total_assigned_events,
  COUNT(*)   FILTER (WHERE NOT e.closed) AS open_events,
  COUNT(*)   FILTER (WHERE     e.closed) AS closed_events,
  COUNT(*)   FILTER (
    WHERE e.closed
      AND e.ts_closed >= now() - INTERVAL '7 days'
  )                                      AS closed_last_week,
  COALESCE(
    CAST(
      ROUND(
        AVG(EXTRACT(EPOCH FROM (e.ts_closed - e.ts)) / 3600),
        2
      )
    AS text)
    || ' h',
    '0.00 h'
  )                                      AS avg_resolution
FROM public.evento AS e
WHERE e.operatore = :IDuser;
```

6 Conclusions

In conclusion, we believe we have laid a solid, real-world foundation for an IDS GUI. Our experience with WebRatio has been very positive: after overcoming the initial learning curve, we were able to implement features rapidly, showcasing the power of no-code development.