Linguaggio Script delle attività IVA

Versione 1.6

BOSCH

Invented for life

Sommario

1 Registro dei cambiamenti

1.1 Versione 1.

La versione 1.0 è integrata nel firmware 3.0 assieme ad IVA 3.0

1.2 Upgrade alla versione 1.1

La versione 1.1 è integrata nel firmware 3.5 assieme ad IVA 3.5:

- Aggiunta la definizione di `ColorHistogram`
- Aggiunto lo stato `SimilarToColor` per confrontare l'istogramma dell'oggetto con quello definito dall'utente
- Aggiunti gli stati `HasObjectSize`, `HasAspectRatio`, `HasVelocity`, `HasDirection`, e `HasColor` per controllare la presenza e l'assenza di proprietà applicate agli oggetti

1.3 Upgrade alla versione 1.2

La versione 1.2 è integrata nel firmware 4.0 assieme ad IVA 4.0:

- Aggiunta la definizione di `FlowDetector`
- Aggiunto lo stato `FlowDetected`
- Aggiunti gli stati `HasFace` e `HadFace`
- Aggiunte le proprietà `FaceWidth` e `MaxFaceWidth`
- Aggiunte le opzioni del campo `ObjectSet` e `SetRelation`

1.4 Upgrade alla versione 1.3

La versione 1.3 è integrata nel firmware 5.0 assieme ad IVA 5.0:

- Aggiunta la definizione di `CrowdDensityEstimator`
- Aggiunto lo stato semplice `EstimatedCrowdDensity`

1.5 Upgrade alla versione 1.4

La versione 1.4 è integrata nel firmware 5.5 assieme ad IVA 5.5

- Aggiunta la definizione di `Counter`

1.6 Upgrade alla versione 1.5

La versione 1.5 è integrata nel firmware 5.6 assieme ad IVA 5.6:

- Aggiunta la definizione di `Resolution`

1.7 Upgrade alla versione 1.6

La versione 1.6 è integrata nel firmware 6.0 assieme ad IVA 6.0:

- Aggiunta la definizione di `MotionDetector`
- Aggiunti gli stati `HasClass` e `HadClass`
- Rimossi gli stati HasFace e HadFace
- Rimosse le proprietà `FaceWidth` e `MaxFaceWidth`

2 Definizioni

2.1 Eventi e stati

Gli stati sono funzioni temporali, invece gli eventi accadono ogniqualvolta uno stato modifica il proprio valore. Il linguaggio script fornisce entrambi, eventi e stati, viste le differenti attività definite per essi. Da un lato, gli eventi permettono di esprimere relazioni temporali tra gli oggetti. Dall'altro, le relazioni non temporali (per esempio spaziali) degli oggetti sono descritte più facilmente dagli stati.

2.2 Proprietà

Per distinguere tra stati Booleani e non-Booleani, è introdotta la nozione di proprietà in riferimento agli stati non-Booleani.

2.3 Regole

Quando si parla di regole, si intendono sia gli eventi sia gli stati che siano output dell'Alarm Task Engine, ossia che sono stati dichiarati esterni. Una regola è definita attiva se il corrispondente stato è attivo o il corrispondente evento è scattato. Nel caso di un evento, la regola corrispondente è attiva a partire dal frame in cui l'evento è scattato fino al successivo frame elaborato in cui la regola è di nuovo inattiva.

2.4 Alarm Task Engine

Alarm Task Engine analizza l'output del modulo VCA (Video Content Analysis) e cerca all'interno del flusso le regole definite secondo il linguaggio script delle attività IVA.

3 Integrazione del sistema

3.1 Panoramica del sistema

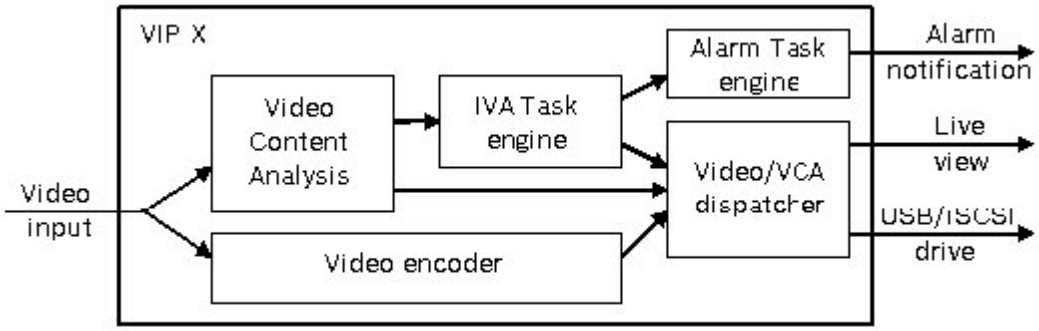

Figure 3.1 VIP X architecture

L'immagine "VIP X architecture" mostra le parti della struttura VIP x pertinenti al Video Content Analysis (VCA). In primis il segnale video è inviato all'encoder video e al software VCA. Il primo invia per esempio flussi H.263 o H.264, il secondo produce un flusso di metadati criptato nel formato Bosch VCD. Entrambi gli output sono ricevuti dal dispatcher video/VCA, che li manda ai client interessati ai dati. Lì una funzione di registrazione è trattata come un altro semplice client.
Il modulo VCA rileva oggetti nella scena indipendentemente da qualsiasi regola definita dall'utente. Gli oggetti estratti e le loro proprietà sono inviati come un flusso VCD all' Alarm Task engine, che rileva le regole definite dall'utente. Gli output dell'Alarm Task engine sono fino a 16 segnalazioni di allarme. Da un lato, sono unite al flusso VCD. Dall'altro lato, IVA Task engine è avvertito di cambiamenti delle segnalazioni d'allarme. IVA Task engine permette di associare qualsiasi tipo di azione alle segnalazioni d'allarme; azioni come inviare e-mail, far scattare un relè o notificare ad un sistema di gestione video.

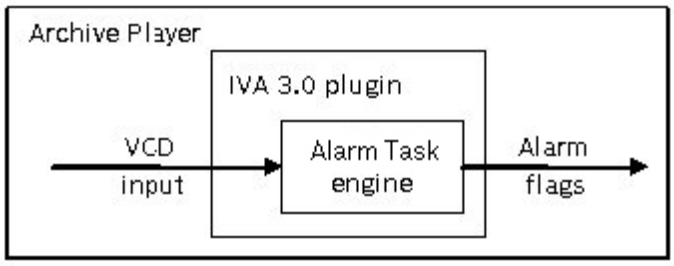

Figure 3.2 Archive Player architecture

Alarm Task engine è configurato con uno script specificato nel linguaggio script delle attività IVA. Il medesimo linguaggio è utilizzato nelle applicazioni di ricerca forense di Bosch (si vede la figura "Archive Player architecture") per cercare eventi imprevisti nelle registrazioni. Lì i flussi VCD registrati sono alimentati con un'altra istanza dell'Alarm Task engine, che può funzionare con un altro gruppo di regole. Le occorrenze degli eventi sono mostrati in una cronologia che

permette una navigazione veloce delle sequenze video rilevanti. Per fornire differenti tipi di strumenti di ricerca forense Alarm Task engine è compresso in un plugin che può essere facilmente scambiato. Il plugin IVA 3.0 è il primo che supporta il linguaggio script delle attività IVA e la configurazione del corrispondente Alarm Task engine.

3.2 Attività definite dall'utente

Tramite il plugin di IVA 3.0 può essere configurato l'Alarm Task engine. Per semplificare la questione, il plugin fornisce una lista onnicomprensiva di query predefinite coi corrispondenti wizard. Gli wizard automaticamente introducono il codice script dell'attività IVA nello script dell'Alarm Task engine. Questi frammenti di codice sono incorniciati da commenti che permettono successivamente la modifica delle attività. La modifica di un codice generato automaticamente può portare ad attività corrotte, che non possono essere più interpretati dai wizard.
Il plugin di IVA 3.0 permette la definizione di un massimo di otto attività, che corrispondono alle prime otto regole esterne dello script. Tuttavia, per rendere visibile una regola definita dall'utente in questa lista di attività, deve essere scritta come segue:

```
// @Task T: 0 V: 0 I : <n> "user defined task"  {
external  ObjectState #<n>  := true;
//@}
```

In tal modo <n> è una variabile metasintattica per l'ID di una regola definita dall'utente. Inoltre non deve esserci nessun'altra attività con lo stesso ID. Il nome dell'attività può essere specificato all'interno delle virgolette.

4 Sintassi

4.1 Tipi

4.1.1 Tipi di base

Il linguaggio script delle attività IVA supporta come tipi di base BOOLEAN, INTEGER, FLOAT, ANGLE e OID. I domini di questi tipi sono i seguenti: BOOLEANs possono avere il valore true o false, INTEGERs sono numeri interi 32-bit con segno, FLOATs rappresentano numeri flottanti a 32-bit, ANGLEs sono specificati in gradi, e OIDs sono gli identificatori unici degli oggetti. Per rappresentare la natura periodica degli angoli, è introdotto il tipo speciale ANGLE. L'unica operazione permessa per gli angoli è un controllo del range. Restituisce true se esiste un angolo equivalente all'interno del range specificato. Due angoli sono equivalenti se la loro differenza è un multiplo di 360 gradi.
Esiste un valore speciale NAN (not a number) che indica un valore che non esiste. Per esempio è restituito NAN quando si accede alla direzione di un oggetto che non esiste più.

Esempio: eccesso di velocità

Il seguente esempio di script di attività IVA controlla se tutti gli oggetti si stiano muovendo con una velocità maggiore di 30 [m/s]; gli oggetti che superano questo limite fanno scattare un allarme:

```
external ObjectState #1 := Velocity within(30.0, *) ;
```

La funzione Velocity restituisce la velocità di un oggetto. L'operazione within prende il corrispondente valore FLOAT e lo confronta con l'intervallo limitato a sinistra (30.0, *). Il valore risultante BOOLEAN è assegnato a ObjectState #1 dell'oggetto considerato.

4.1.2 Attributi

Gli eventi e gli stati possono essere estesi con gli attributi. Ciascun attributo è caratterizzato da un tipo di base e da un nome unico. Il nome è necessario per distinguere gli attributi dello stesso tipo e per accedere all'attributo. Quando si introducono gli eventi, la seguente sintassi sarà utilizzata per descrivere gli attributi:

```
<event> -> ( <type> : <name>, … , <type> : <name> )
```

Per gli stati è utilizzata una sintassi similare:

```
<state>( <type> : <name>, … , <type> : <name> ) -> <type>
```

La lista degli attributi può essere vuota in entrambi i casi.

Esempio: attraversamento di due linee

Lo script dell'attività IVA seguente definisce due linee e fa scattare l'Event #1 se il medesimo oggetto attraversa la prima linea e subito dopo la seconda. Per controllare che entrambi gli eventi CrossedLine siano stati attivati dallo stesso oggetto, sono confrontati gli attributi oid di entrambi gli eventi. Senza questa condizione, Event #1 sarebbe attivato anche nel caso in cui un oggetto abbia attraversato Line #2 dopo che uno completamente diverso abbia attraversato Line #1. Le parole chiave first e second permettono l'accesso alle liste di attributi dei due eventi coinvolti in una relazione before.

```
Line #1 := {
    Point (10, 10) Point (10,50) Direction (0)
};
Line #2 := {
   Point (50,10) Point (50,50) Direction (0)
};
external Event #1 := {
        CrossedLine #1 before CrossedLine #2
        where first.oid == second.oid
};
```

4.2 Commenti

Il linguaggio script delle attività IVA permette di effettuare commenti come nel linguaggio C. Questo significa che tutti i caratteri di una riga dopo due slash / / sono ignorati dall' Alarm Task engine, così come tutti i caratteri incorniciati da / * (slash ed asterisco) e * / (asterisco e slash). L'ultimo modo può essere utilizzato per commentare molteplici righe in una volta o una parte di una singola riga.

```
Field #1 := { // questo è un commento
     Point (10, 10)  / *Point (10, 50) */ Point (50, 50)
     Point (50, 10)
};
/ * tutti i caratteri
    all'interno di questo paragrafo
    sono commenti * /
```

4.3 Normalizzazione degli script

Il linguaggio script delle attività IVA fornisce un metodo per normalizzare le coordinate all'interno di un range specificato. Si ponga la parola chiave `Resolution` all'inizio dello script. La coordinata `Min` definisce l'angolo superiore a sinistra dell'immagine e la coordinata `Max` definisce l'angolo inferiore destro. Tutte le successive coordinate sono poi definite in relazione a `Min` e `Max`.

Esempio

```
Resolution := { Min {0, 0} Max {1,1} };
Line #1 := {
    Point (0.10, 0.10) Point (0.10, 0.50)
    Point (0.50, 0.50) Point (0.50, 0.10)
};
```

4.4 Primitive

Il linguaggio script delle attività IVA prevede le primitive geometriche. Queste primitive tengono sotto osservazione singoli oggetti e fanno scattare eventi con determinate azioni. Il numero delle primitive che può essere inizializzato è limitato dalla memoria del dispositivo su cui sta lavorando l'Alarm Task engine. Nella versione firmware 3.0 questo limita il numero dei percorsi ad otto, ed il numero totale di primitive a 32.

4.4.1 Campo

Sintassi

Le primitive `Field` sono definite nella maniera seguente:

```
Field #<n> := {
   Point (<x>, <y>) … Point (<x>, <y>)
   [DebounceTime (<time>)]
   [ObjectSet (<objectset>)]
   [SetRelation (<relation>)]
};
```

<n> è il numero del campo e deve essere compreso tra 1 e 32. I punti specificati formano un poligono. Il poligono deve essere formato da un minimo di 3 ad un massimo di 16 punti e deve essere semplice, ossia non deve intersecarsi con se stesso. Le coordinate dei punti sono specificate in pixel con la risoluzione dell'immagine elaborata dall'algoritmo di IVA. `<time>` specifica il `DebounceTime` opzionale in secondi. Il suo valore di default è 0. Quando è stato configurato un `DebounceTime`, lo stato di un oggetto, che sia dentro o fuori dal campo, si modifica soltanto se l'oggetto sta dall'altra parte della linea per almeno il tempo specificato da `<time>`. In questo modo si possono eliminare gli errori di posizione nel tracking di un oggetto o gli allarmi multipli di oggetti che si muovono lungo il margine di un campo.
Se è utilizzato un rilevatore di flusso, il `DebounceTime` ha un significato differente. `<time>` specifica il tempo di post-allarme. Questo garantisce che, se sono rilevati molti piccoli allarmi di flusso all'interno del debounce time `<time>`, sono uniti in un lungo periodo di allarme.
`ObjectSet` e `SetRelation` specificano quali parti dell'oggetto sono considerate per determinare se sia ritenuto dentro o fuori dal campo. In tal modo `<objectset>` può essere configurato a `BaryCenter` o `BoundingBox` con `BaryCenter` come valore di default. `<relation>` può assumere i valori `Intersection` e `Covering`, con `Intersection` come valore di default. Per esempio, se `BoundingBox` e `Covering` sono selezionate, il parallelogramma virtuale dell'oggetto deve essere completamente all'interno del poligono specificato per essere considerato all'interno. Se `<objectset>` è fissato a `BaryCenter`, entrambe le opzioni di `<relation>` sfociano nello stesso comportamento, dato che l'oggetto fissato consiste solo di un unico punto.

Stati

```
InsideField #<n> (OID: oid)  -> BOOLEAN
```

Lo stato `InsideField #<n>` è fissato quando l'oggetto `oid` è dentro il `Field #<n>`.

```
ObjectsInField #<n>  -> INTEGER
```

`ObjectsInFIeld #<n>` restituisce il numero di oggetti attualmente nel `Field #<n>`.

Eventi

```
EnteredField #<n>  -> (OID: oid)
```

L'evento `EventField` è messo in moto quando un oggetto entra nel `Field #<n>`. L'oggetto che ha causato l'evento può essere interrogato dall'argomento `oid`.

```
LeftField #<n> -> (OID: oid)
```

L'evento `LeftField` è azionato quando un oggetto lascia il `Field #<n>`. L'oggetto che ha causato l'evento può essere interrogato dall'attributo `oid`.
Si noti che un oggetto non aziona il corrispondente evento `EnteredField #<n>` quando è già all'interno di un campo al primo rilevamento.

Esempio

L'esempio seguente fa scattare un allarme se lo stesso oggetto è entrato nel campo specificato e successivamente ne è uscito di nuovo.

```
Field #1 := {
    Point (10, 10)  Point (10, 50)
    Point (50, 50)  Point (50,10)
};
external Event #1 := {
    EnteredField #1 before LeftField #1
    where first.oid == second.oid
};
```

4.4.2 Linea

Sintassi

Le primitive `Line` sono definite nel seguente modo:

```
Line #<n> := {
    Point (<x>, <y>)  Point (<x>, <y>)
    Direction (<dir>)
    [DebounceTime (<time>)]
};
```

<n> è il numero della linea e deve essere compreso tra 1 e 16. Una linea è costituita da esattamente due punti, le cui coordinate sono specificate in pixel con la risoluzione dell'immagine elaborata dall'algoritmo di IVA. Con l'argomento <dir> si può scegliere se un oggetto che attraversa la linea azioni l'evento o se solo gli oggetti che attraversano da destra a sinistra o nella direzione opposta siano rilevanti. Nel primo caso, si prevede che <dir> sia pari a 0. Negli ultimi casi, <dir> prevede il valore 1 e 2 rispettivamente. <time> specifica il DebounceTime opzionale. Quando è fissato il DebounceTime, un evento CrossedLine #<n> è azionato soltanto se lo stesso oggetto non attraversa subito dopo la medesima linea nella direzione opposta, all'interno della finestra temporale specificata in <time>. In questo modo si eliminano gli errori di posizione nel tracking degli oggetti o gli allarmi multipli di oggetti che si muovono lungo la linea.

Eventi

```
CrossedLine #<n>  -> (OID: oid)
```

L'evento CrossedLine #<n> è azionato quando un oggetto attraversa la Line #<n> nella maniera specificata. L'oggetto che ha causato l'evento può essere interrogato dall'attributo oid.

Esempio

Si veda la sezione "Esempio: attraversamento di due linee" a pagina 10.

4.4.3 Percorso

Sintassi

Le primitive Route sono definite nel modo seguente:

```
Route #<n> := {
    Point (<x>, <y>)  Distance (<r>)
    …
    Point (<x>, <y>)  Distance (<r>)
    Direction (<dir>)
    MinPercentage (<min>)
    MaxGap (<max>)
};
```

<n> è il numero del percorso e deve essere compreso tra 1 e 32. Un percorso è caratterizzato da almeno due punti fino ad un massimo di otto. Le coordinate del punto sono specificate in pixel con la risoluzione dell'immagine elaborate dall'algoritmo di IVA. Ciascun punto è seguito da una tolleranza <r>. Con queste tolleranze il percorso dei punti si allarga fino a diventare una striscia. Gli oggetti che si muovono lungo la striscia nella direzione specificata azionano l'evento `FollowedRoute #<n>`. Se <dir> è impostato al valore 1, sono considerati solo i movimenti dell'oggetto che vanno dal primo punto verso l'ultimo. Se <dir> è impostato al valore 2, sono considerati solo i movimenti dell'oggetto che vanno nella direzione opposta. Se <dir> è uguale a 0, è preso in considerazione qualsiasi movimento dell'oggetto all'interno della striscia. I movimenti dell'oggetto che non soddisfano il limite direzionale sono ignorati.
I parametri `MinPercentage` e `MaxGap` specificano la tolleranza del rilevatore. Se `Direction` è configurata a 0, il significato dei due parametri è il seguente. Il rilevatore ricorda per ciascun oggetto quali parti della striscia sono state solcate. Se è stata solcata più della `MinPercentage` dell'intera striscia e il salto più grande tra due parti solcate (compresi i salti all'inizio e alla fine della striscia) è inferiore a `MaxGap`, allora l'evento `FollowedRoute #<n>` è azionato. Se è stata assegnata una direzione al percorso, i movimenti dell'oggetto all'interno del percorso sono presi in considerazione soltanto se il movimento corrisponde alla direzione specificata e se la distanza dall'ultima parte solcata non è più grande di `MaxGap`.

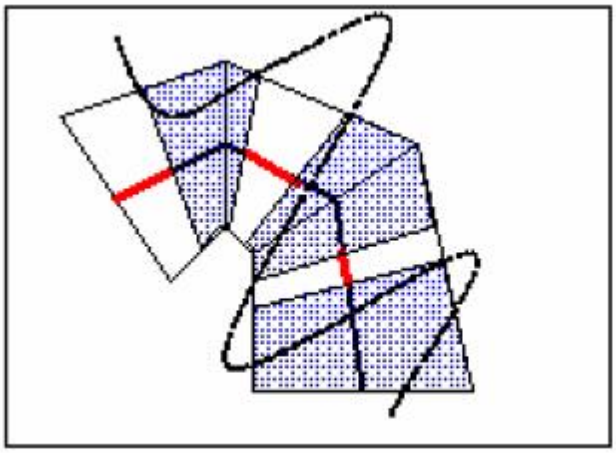

Figure 4.1 Illustration of an object following a predefined route

La figura in alto mostra un oggetto che segue un percorso predefinito. Le parti del percorso che sono state solcate dall'oggetto sono segnate in blu. I salti nel mezzo sono sottolineati con una linea rossa.

Eventi

```
FollowedRoute #<n> -> (OID:oid)
```

L'evento `FollowedRoute #<n>` è azionato quando un oggetto ha seguito il percorso #<n>. L'oggetto che ha causato l'evento può essere interrogato dall'attributo `oid`.

Esempio

```
Route  #1 := {
    Point (20,20)  Distance (5)
    Point (20,50)  Distance (5)
    Point (50,50)  Distance (5)
    Direction (0)
    MinPercentage (90)
    MaxGap (20)
};
external Event  #1 := FollowedRoute  #1;
```

4.4.4 Stazionare in modo sospetto

Sintassi

Le primitive `Loitering` sono definite nel modo seguente:

```
Loitering #<n>  := {
    Radius (<r>)
    Time (<time>)
};
```

<n> è il numero della primitiva `Loitering` e deve essere compreso tra 1 e 32. La primitiva `Loitering` rileva gli oggetti che stanno in un unico posto per `<time>` secondi. <r> specifica la tolleranza spaziale in metri del rilevatore di loitering. Per misurare i movimenti dell'oggetto in metri, la telecamera deve essere stata precedentemente calibrata.

Stati

```
IsLoitering #<n> (OID:oid)  -> BOOLEAN
```

Lo stato di un oggetto `oid IsLoitering  #<n>` è fissato quando l'oggetto staziona nel medesimo luogo per almeno il tempo specificato.

Esempio

```
Loitering #1 := {
    Radius (5)
    Time (10)
};
external ObjectState #1  := IsLoitering #1;
```

4.4.5 ColorHistogram

Sintassi

Le primitive `ColorHistogram` sono definite nel modo seguente:

```
ColorHistogram #<n> := {
    HSV ( <h>, <s>, <v> [, <weight>] )
    … HSV (<h>, <s>, <v> [, <weight>] )
    Similarity (<similarity> )
    Outlier (<outlier>)
};
```

<n> è il numero della primitiva ColorHistogram e deve essere compreso tra 1 e 32. La primitiva `ColorHistogram` confronta i colori degli oggetti con i colori definiti dall'utente. Fino a cinque colori di base possono essere utilizzati con la parola chiave HSV. Il valore di default del parametro opzionale `<weight>` è 1. Il peso totale dei colori di base non deve eccedere 255. I colori di base sono definiti nello spazio di colore HSV con <h> (un valore tra 0 e 360), che rappresenta la tonalità, <s> (un valore tra 0 e 100) che rappresenta la saturazione, e <v> (un valore tra 0 e 100) che indica l'intensità. La figura "HSV cone" mostra i tre componenti dello spazio di colore HSV.

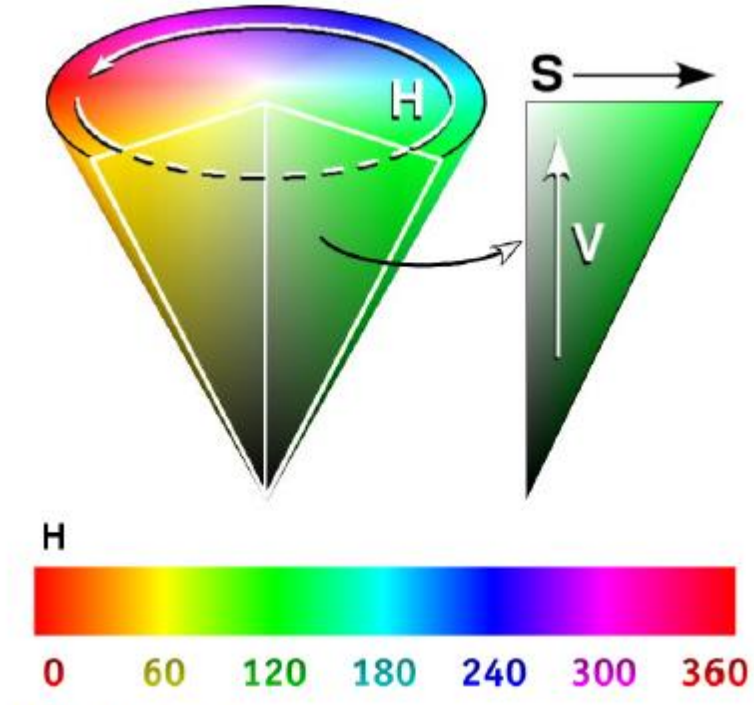

Figure 4.2 HSV cone

`<similarity>` è un valore compreso tra 0 e 100 e specifica quanto debba essere simile un istogramma di colore per essere considerato come corrispondente. Più sono simili due istogrammi più è grande la loro

<similarity>. Il parametro <outlier> permette corrispondenze parziali tra i colori definiti dall'utente e l'istogramma cromatico dell'oggetto, ossia i colori definiti dall'utente possono coprire solo una sezione dell'istogramma cromatico dell'oggetto e i restanti colori dovrebbero essere considerati come outlier. Si consideri il seguente esempio: quando si cercano persone che indossano una giacca rossa, circa il 50% dei colori dell'oggetto dovrebbe essere rosso e l'altro 50 % non viene preso in considerazione.

Stati

```
SimilarToColor #<n>  (OID:oid)  -> BOOLEAN
```

Lo stato SimilarToColor #<n> di un oggetto oid è fissato quando il più recente istogramma cromatico dell'oggetto è almeno simile all'istogramma cromatico definito dall'utente quanto la soglia considerata.

Esempio

Il seguente istogramma cromatico rileva oggetti che presentano almeno colori rossastri al 25% e colori scuri almeno al 25%.

```
ColorHistogram #1 := {
    HSV (0, 100, 100)
    HSV (0, 0, 0)
    Similarity (90)
    Outliers (50)
};
external ObjectState #1 := SimilarToColor #1;
```

4.4.6 FlowDetector

Sintassi

Le primitive FlowDetector possono essere definite in uno dei seguenti modi:

```
FlowDetector #<n> := {
    [Direction (<minangle>, <maxangle>)]
    [Direction (<minangle>, <maxangle>)]
    [Velocity (<minvelocity>, <maxvelocity>)]
    [Activity (<minactivity>, <maxactivity>)]
    [Field #<m>]
};
```

Oppure:

```
FlowDetector #<n>  := {
    CounterFlow (<timewindow>, <angletol>)
    [Velocity (<minvelocity>, <maxvelocity>)]
    [Activity (<minactivity>, <maxactivity>)]
    [Field #<m>]
};
```

Il rilevatore di flusso funziona grazie al campo di flusso significativo calcolato dagli algoritmi di IVA. Ciascun vettore di flusso rilevato deve soddisfare un certo numero di filtri definiti dall'utente per azionare l'allarme. Nella prima definizione possono essere indicati fino a due filtri direzionali. Se le due direzioni sono specificate, un vettore di movimento deve soddisfarne almeno una per un'elaborazione ulteriore. Ciascun filtro direzionale è composto da un `<minangle>` e un `<maxangle>` specificati in gradi. Il sistema di coordinate per gli angoli è mostrato nella figura "Image coordinates".
Nella seconda definizione la direzione è determinata automaticamente in base all'analisi del flusso principale. Sono considerati gli ultimi `<timewindow>` secondi per calcolare la direzione del flusso. Non appena è rilevata una direzione principale, essa definisce il flusso principale ed attiva il rilevatore di flusso. Se non c'è una direzione dominante all'interno degli ultimi `<timewindow>` secondi, il rilevatore di flusso è inattivo. Un vettore di movimento che si muove nella direzione opposta al flusso principale con una tolleranza angolare massima di `<angletol>` gradi supera il filtro direzionale della seconda definizione. Se è specificato un limite spaziale, solo i vettori di movimento all'interno del `Field #<m>` sono presi in considerazione quando si valuta la direzione del flusso principale. Gli altri filtri non si applicano alla stima del flusso principale.
Possono essere configurati filtri supplementari per la velocità, l'attività e lo spazio. Nell'esempio `<minvelocity>` e `<maxvelocity>` sono specificati in pixel al secondo. Se un `Field #<m>` è aggiunto alla definizione, i vettori di movimento fuori dal `Field #<m>` sono ignorati durante l'elaborazione. Se non viene aggiunto alcun campo, tutti i vettori di movimento sono considerati. L'attività misura il numero di vettori di movimento attivi, ossia il numero di vettori di movimento che abbiano superato tutti i filtri. In questo caso l'attività è 0 se nessun vettore di movimento è attivo. L'attività massima, pari a `100`, è raggiunta quando il campo specificato è completamente pieno di vettori di movimento attivi. Con `<minactivity>` e `<maxactivity>` l'utente può definire un intervallo di attivazione all'interno del quale il rilevatore di flusso aziona un allarme. In tal modo il limite inferiore `<minactivity>` è escluso dall'intervallo. Questo implica che il rilevatore di flusso azionerà un allarme solo se è presente del movimento.
Se il `Field #<m>` è presente nella definizione di rilevatore di flusso, il rilevatore eredita il `DebounceTime` del campo specificato. In questo caso il significato di `DebounceTime` è che il campo di flusso deve superare tutti i limiti per almeno `DebounceTime` secondi prima che il rilevatore di flusso azioni un allarme.

Stati

```
DetectedFlow #<n>   -> BOOLEAN
```

Lo stato DetectedFLow #<n> è fissato quando è presente un significativo flusso che rispetti tutti i limiti definiti nella corrispondente definizione di FlowDetector #<n>.

Esempio

Il seguente esempio fa scattare un allarme se un movimento significativo da destra a sinistra è stato rilevato all'interno del rettangolo specificato.

```
Field #1  := {
    Point (10,10)  Point (10,50)
    Point (50, 50)  Point (50,10)
};
FlowDetector #1  := {
Direction (-45, 45)
Field #1
};
external SimpleState #1 := DetectedFlow #1;
```

4.4.7 CrowdDensityEstimator

Sintassi

Le primitive CrowdDensityEstimator sono definite nel modo seguente:

```
CrowdDensityEstimator #<n>  := {
  [Activity (<minactivity>, <maxactivity>)]
  CrowdDensityField #n
  [DebounceTime (<seconds>) | SmoothingTime (<seconds>)]
};
```

Il rilevatore che stima il livello di folla utilizza un'immagine di riferimento, che dovrebbe mostrare la scena vuota, in modo da rilevare la folla di fronte allo sfondo di riferimento. Questa scena è limitata dal CrowdDensityField, che deve essere specificato in anticipo. L'algoritmo di IVA calcola un valore di attività del livello di folla per la regione data, che va da zero a 100%. Il livello di attività di soglia può essere limitato fissando l'intervallo di attivazione tra <minactivity> e <maxactivity>. Il livello di attività di folla è calcolato ogni secondo. Per ignorare un salto temporaneo del livello di folla possono essere configurati i filtri DebounceTime e SmoothingTime. Se è utilizzato DebounceTime, un allarme è azionato se il livello di folla è all'interno delle soglie di attività per più che il tempo specificato. Con SmoothingTime <seconds> definisce una finestra variabile: un allarme è azionato se la media nel tempo specifico è all'interno dei limiti di attività.

Stati

```
EstimatedCrowdDensity #<n>  -> BOOLEAN
```

Lo stato EstimatedCrowdDensity #<n> è fissato quando è presente un livello di folla significativo, che soddisfi i limiti definiti nella corrispondente definizione CrowdDensityEstimator #<n>.

Esempio

Il seguente esempio aziona un allarme se il livello di folla è maggiore o uguale al 25% per più di dieci secondi all'interno del campo di folla, che deve essere specificato e salvato sul dispositivo prima della query o prima della registrazione.

```
CrowdDensityEstimator #1  := {
    Activity (25,100)
    CrowdDensityField #1
    DebounceTime (10)
};
external SimpleState #1 := EstimatedCrowdDensity #1;
```

4.4.8 MotionDetector

Sintassi

Le primitive MotionDetector sono definite nel modo seguente:

```
MotionDetector  #<n>  := {
    [Activity (<minactivity>, <maxactivity>)]
    [Size (<minsize>, <maxsize>)]
    [Field #<n>]
};
```

Il rilevatore di movimento fa scattare un allarme se le celle di movimento fornite soddisfano l criteri definiti. MotionDetector può operare sullo schermo intero o sul Field specificato. Il livello di attività di soglia può essere limitato configurando l'intervallo di attivazione tra <minactivity> e <maxactivity>. L' Activity è specificata in percentuale dell'area selezionata. Inoltre il livello di attività di soglia può essere limitata da Size del gruppo di celle. Questo intervallo (<minsize>, <maxsize>) è specificato in percentuali dello schermo intero.

Stati

```
DetectedMotion #<n>  -> BOOLEAN
```

Lo stato DetectedMotion #<n> è fissato quando è presente un livello di movimento significativo, tale da soddisfare i limiti definiti nella corrispondente definizione di MotionDetector #<n>.

Esempio

Il seguente esempio aziona un allarme se la dimensione del gruppo di celle è maggiore o uguale al 0.5% dell'intero schermo.

```
MotionDetector #1  := {
    Size (0.5, 100)
};
external SimpleState #1  := DetectedMotion #1;
```

4.5 Eventi specifici degli oggetti

La funzione principale dell'algoritmo di IVA è la rilevazione degil oggetti e il loro tracciamento. Oltre alla posizione e alle altre proprietà degli oggetti rilevati in tempo reale, l'algoritmo notifica gli eventi di base degli oggetti. Quando l'algoritmo rileva un nuovo oggetto, un evento `Appeared` è azionato. In modo corrispondente un evento `Disappeared` è inviato non appena un oggetto è perso. `Idled` e `Removed` sono casi speciali di oggetti che scompaiono, con il seguente significato. Se un oggetto non si muove assolutamente per un certo intervallo di tempo, un evento `Idled` è azionato. Questo succede se una persona abbandona un oggetto come una borsa. In modo simile, un oggetto può essere preso da una persona azionando l'evento `Removed`.

4.5.1 Eventi degli oggetti

Gli eventi degli oggetti sono azionati da oggetti apparsi o scomparsi.

Eventi

```
Appeared (OID:oid)
Disappeared (OID:oid)
Idled (OID:oid)
Removed (OID:oid)
```

4.6 Contatore

Sintassi

Un contatore è definito in uno dei modi seguenti:

```
Counter #<n>  := {
    Event #<m>
};
```

oppure:

```
Counter #<n>  := {
    Counter #<x> + Counter #<y>
};
```

oppure:

```
Counter #<n>  := {
    Counter #<x> - Counter #<y>
};
```

oppure:

```
external Counter #<n>  := {
…
};
```

Il contatore può contare il numero degli eventi azionati oppure al contatore può essere attrivuita la somma o la differenza di due contatori. Inoltre la parola chiave `external` indica che il valore del contatore è aggiunto al flusso VCD così come è incluso nel messaggio di conteggio del RCP. Di conseguenza senza questa parola chiave i contatori sono utilizzati soamente internamente e il valore attuale non può essere mostrato. Fino a 32 contatori possono essere configurati. Il range di default di ciascun contatore è tra 0 e il valore massimo di un 32-bit.
I seguenti argomenti facoltativi possono essere configurati: ciascun contatore può essere etichettato tramite `Text ("<string>")` in cui la stringa è a 32-bit e codificata UTF-8. Il contatore prevede due differenti modalità – `Mode (KeepMax)` oppure `Mode (Wraparound)`. Se la modalità `KeepMax` è configurata, il contatore si ferma al limite superiore. Nella modalità `Wraparound` il contatore riparte dal limite inferiore (per esempio `0`). I limiti inferiori e superiori possono essere decisi con l'argomento `within (<min>,<max>)`, in cui `<min>` corrisponde al limite inferiore e `<max>` al limite superiore. La posizione del valore del contatore mostrato nel video può essere definita con la parola chiave `TopLeft (<x>, <y>)`, che indica la posizione all'interno dell'immagine. L'origine è nell'angolo superiore a sinistra.
Generalmente un contatore non aziona un allarme. Per far si che scatti, è necessario che al contatore sia assegnato uno stato come mostrato sotto:

```
external SimpleState #<n>:= Counter #<n> within(<min>,<max>)
```

Pertanto un allarme è azionato non appena il valore del conteggio è all'interno dei limiti definiti da `<min>` e `<max>`.

Esempio

```
ObjectState #32  := true;
Event #2  := OnSet ObjectState #32;
external Counter #5  := {
    Event #2 Text ("Every 5th:")
    TopLeft (4, 14) within(0,5)  Mode (Wraparound)
};
external SimpleState #2  := Counter #5 within (5, *);
```

4.7 Proprietà degli oggetti

Ciascun oggetto tracciato ha una serie di proprietà. Queste proprietà comprendono la posizione dell'oggetto, la direzione del suo movimento, la sua velocità, la sia dimensione, il suo bounding box. Mentre la posizione è accessibile solo indirettamente tramite primitive geometriche, le altre proprietà invece sono disponibili direttamente nelle espressioni condizionali tramite le funzioni seguenti. Tutte le funzioni di seguito restituiscono NAN se l'oggetto non possiede quella proprietà o se l'oggetto non esiste affatto. Per ciascuna proprietà esiste uno stato menzionato sotto che restituisce `true` se la proprietà è presente e `false` nel caso contrario. Per tutti gli stati e le funzioni seguenti, oid è opzionale se la lista degli attributi del'obiettivo corrente contiene l'argomento `OID:oid`.

Funzioni

```
Direction (OID:oid)  -> ANGLE
```

La funzione restituisce la direzione attuale in gradi dell'oggetto `oid`. Se nessuna direzione è disponibile per quest'oggetto, il risultato è NAN. Le direzioni sono espresse in coordinate dell'immagine. In questo modo un movimento dal bordo destro dell'immagine verso il bordo sinistro corrisponde a zero gradi, dal bordo superiore a quello inferiore corrisponde a 90 gradi, dal limite sinistro a quello destro a corrisponde a 180 gradi, e infine dal bordo inferiore a quello superiore a 270 gradi (si guardi la figura "Image coordinates").

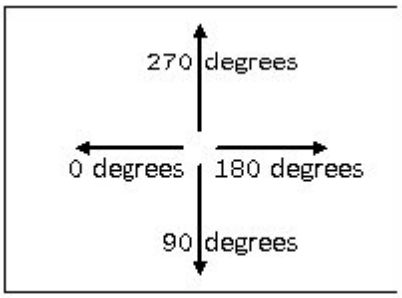

Figure 4.3 Image coordinates

```
Velocity  (OID:oid)  -> FLOAT
```

Questa funzione restituisce la velocità attuale in [m/s] dell'oggetto `oid`. La velocità è stimata utilizzando la traslazione dell'oggetto in coordinate ed utilizzando i parametri di calibrazione della telecamera. Se non è disponibile alcuna velocità per quest'oggetto, il risultato è NAN.

```
AspectRatio  (OID:oid)  -> FLOAT
```

Questa funzione restituisce le proporzioni attuali dell'oggetto `oid`. E' definita come il rapporto tra l'altezza e la larghezza del bounding box dell'oggetto. Se non è disponibile alcun bounding box per quest'oggetto, il risultato è NAN. Un

quadrato ha una proporzione pari ad 1. Un oggetto due volte più alto che largo ha una proporzione pari a 2 (si veda la figura “Current aspect ratio”).

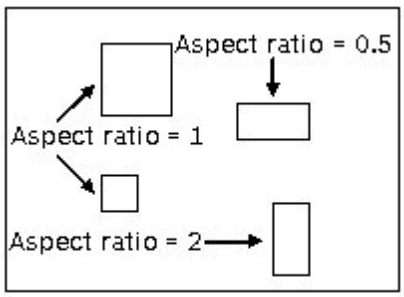

Figure 4.4 Current aspect ratio

```
ObjectSize  (OID:oid)
```

Questa funzione restituisce la dimensione attuale in [m²] dell’oggetto `oid`. La dimensione è stimata in base alla forma dell’oggetto e ai parametri di calibrazione della telecamera. Se non è disponibile alcuna dimensione per l’oggetto, il risultato è NAN.

4.7.1 Stati

```
HasDirection (OID:oid)  -> BOOLEAN
```

Restituisce `true` se la direzione dell'oggetto con l'argomento `ID` è disponibile nell'immagine attuale. Se l'oggetto non è presente nel frame attuale il valore restituito è `false`.

```
HasVelocity (OID:oid)  -> BOOLEAN
```

Restituisce `true` se la velocità dell'oggetto con l'argomento `ID` è disponibile nell'immagine attuale. Se l'oggetto non è presente nel frame attuale il valore restituito è `false`.

```
HasAspectRatio (OID:oid)  -> BOOLEAN
```

Restituisce `true` se il bounding box dell'oggetto con l'argomento `ID` è disponibile nell'immagine attuale. Se l'oggetto non è presente nel frame attuale il valore restiuito è `false`.

```
HasObjectSize (OID:oid)  -> BOOLEAN
```

Restituisce `true` se il bounding box dell'oggetto con l'argomento `ID` è disponibile nell'immagine attuale. Se l'oggetto non è presente nel frame attuale il valore restiuito è `false`.

```
HasColor (OID:oid)  -> BOOLEAN
```

Restituisce `true` se l'istogramma cromatico dell'oggetto con l'argomento `ID` è disponibile dall'ultima apparizione dell'oggetto. Se esso non è presente nel frame attuale il valore restituito è `false`.

```
HasClass (<classnames>)  -> BOOLEAN
```

Restituisce `true` se è rilevato nel frame attuale un oggetto che corrisponde ad una delle classi specificate. Se un oggetto del genere non è presente nel frame attuale, il valore restituito è `false`.

```
HadClass (<classnames>)  -> BOOLEAN
```

Restituisce `true` se un oggetto che corrisponde ad una delle classi specificate è stato rilevato dall'ultima apparizione dello stesso. Se un oggetto del genere non è presente nel frame attuale, il valore restituito è `false`.
Per gli ultimi due stati le classi sono definite inserendo i necessari `<classnames>`, presi dalla lista seguente e separati da uno spazio vuoto:

- `Person` (persone che si muovono in piedi)
- `Bike` (biciclette in moto)
- `Car` (macchine in moto)
- `Truck` (camion/grosse macchine in moto)
- `NoClass` (nessuna delle classi di sopra)

4.8 Stati di manomissione

Gli algoritmi di IVA possono rilevare tentativi di manomissione delle telecamere. L'output del rilevamento di una manomissione è disponibile tramite i seguenti stati Booleani:

```
SignalTooNoisy  -> BOOLEAN
```

Se il segnale video diventa troppo rumoroso, tale che il rilevamento automatico dell'oggetto diventa impossibile, lo stato `SignalTooNoisy` è attivato. Questo può accadere se un segnale video analogico è trasmesso su grandi distanze o se la sensibilità del sensore della telecamera non è sufficiente nelle applicazioni di visione notturna.

```
SignalTooDark  -> BOOLEAN
```

Se il segnale video diventa troppo scuro, tale che il rilevamento automatico dell'oggetto diventa impossibile, lo stato `SignalTooDark` è attivato. Questo può succedere se la telecamera è coperta da uno strato, tale che vengano registrate immagini quasi nere.

```
SignalTooBright  -> BOOLEAN
```

Se il segnale video diventa troppo luminoso, tale che il rilevamento automatico dell'oggetto diventa impossibile, lo stato `SignalTooBright` è attivato. Questo può succedere se la telecamera è abbagliata da una forte fonte luminosa.

```
SignalLoss  -> BOOLEAN
```

Se il segnale video è perso, lo stato `SignalLoss` è attivato.

```
GlobalChange  -> BOOLEAN
```

Se la maggior parte del contenuto dell'immagine è cambiato, lo stato `GlobalChange` è attivato. Questo può succedere se la telecamera è spostata o se l'oggetto arriva troppo vicino alla telecamera.

```
RefImageCheckFailed  -> BOOLEAN
```

Se un'immagine di riferimento è stata stabilita durante la configurazione dell'algoritmo di IVA e il controllo del riferimento dell'algoritmo IVA è stato attivato, IVA rileva manipolazioni della telecamera confrontando il segnale video attuale con l'immagine di riferimento prestabilita. Discrepanze significative tra le due immagini sono registrate nel flusso VCD. Questa informazione è accessibile tramite il linguaggio script delle attività iVA con lo stato `RefImageCheckFailed`.

Esempio

Lo script dell'attività IVA seguente aziona un allarme per ogni azione di manonissione rilevata.

```
external SimpleState #1  :=
    SignalTooNoisy or SignalTooDark or SignalTooBright or
    SignalLoss or GlobalChange or RefImageCheckFailed;
```

Si noti che in quest'esempio è importante lavorare con un `SimpleState` definito dall'utente invece che un `ObjectState`, dato che quest'ultimo è valutato soltanto per gli oggetti rilevati. Tuttavia, quando la telecamera è manomessa generalmente non ci sono rilevamenti.

4.9 Stati definiti dall'utente

Gli stati definiti dall'utente sono sempre stati Booleani, ossia questi stati assumono i valori `false` o `true`. Uno stato definito dall'utente o non ha attributi oppure ha l'`ID` dell'oggetto come attributo. Il primo stato è chiamato `SimpleState`, il secondo invece `ObjectState`. Nelle sezioni seguenti è spiegata maggiormente in dettaglio la loro sintassi e il loro utilizzo.

4.9.1 SimpleState

Un `SimpleState` è definito nel modo seguente:

```
[external]  SimpleState  #<n>  := <condition>;
```

In questo caso `<condition>` è la variabile metasintattica di un'espressione Booleana. L'utente può specificare fino a 32 stati (da 1 a 32) tramite il numero <n>, ma solo i primi 16 stati (<n> da 1 a 16) possono essere `external`. Lo stesso `SimpleState` non può essere definito due volte. La lista degli attributi di un `SimpleState` è sempre vuota. Pertanto non si può accedere alle proprietà specifiche dell'oggetto nella proposizione condizionale di un `SimpleState`.
La seguente tabella riassume gli stati predefiniti che possono essere utilizzati al posto della variabile `<condition>` oltre ad un `SimpleState` definito precedentemente:

```
SignalTooNoisy
SignalTooDark
SignalTooBright
SignalLoss
GlobalChange
RefImageCheckFailed
```

4.9.2 Combinazione Booleana delle condizioni

Possono essere combinate molteplici condizioni. La sintassi delle condizioni coordinate è la seguente:

```
<condition>  := <condition> and <condition>

<condition>  := <condition> && <condition>
```

In modo similare le condizioni disgiuntive sono scritte così:

```
<condition>  := <condition> or <condition>

<condition>  := <condition> || <condition>
```

La negazione delle condizioni è la seguente:

```
<condition>  := not <condition>

<condition>  := !<condition>
```

Le ambiguità nella valutazione delle espressioni sono risolte dalle precedenze. Le negazioni hanno la massima priorità, seguite dalle condizioni coordinate, e le disgiunzioni per ultime. Inoltre la priorità può essere controllata utilizzando sottoespressioni con le parentesi:

```
<condition>  := ( <condition> )
```

4.9.3 Proprietà nelle condizioni

Le proprietà e gli stati non-Booleani possono essere utilizzati come condizioni nel modo seguente:

```
<condition>  := <num-expr> within (<min>, <max>)
```

La parola chiave `within` controlla se la specificata `<num-expr>` sia all'interno dell'intervallo indicato. I limiti dell'intervallo, `<min>` e `<max>`, sono valori costanti. Se uno dei due limiti è sostituito dall'asterisco `*`, il limite corrispondente è ignorato. Una `<num-expr>` può essere un attributo, un valore costante, uno stato non-Booleano o una proprietà. In combinazione con la parola chiave within, `<num-expr>` deve essere di tipo `INTEGER`, `FLOAT` oppure `ANGLE`. Oltre alla relazione `within`, è consentita un'equazione semplice per ogni tipo non-Booleano purchè entrambi gli operandi siano dello stesso tipo:

```
<condition>  := <num-expr> == <num-expr>

<condition>  := <num-expr> != <num-expr>
```

`ObjectsOnScreen` è l'unico stato non-Booleano che può essere utilizzato nella definizione di un `SimpleState #<n>`. Restituisce il numero di oggetti che sono attualmente rilevati dall'algoritmo di VCA.
Si noti che le condizioni descritte in questa sezione restituiscono `false` se il valore di una `<num-expr>` è `NAN`.

Esempio

Il seguente script di attività IVA aziona un allarme se almeno due oggetti sono rilevati dall'algoritmo di VCA.

```
external SimpleState #1  := ObjectState within (2,*);
```

L'esempio seguente aziona un evento se due oggetti diversi attraversano la stessa linea. In questo caso il secondo oggetto deve attraversare la linea entro dieci secondi dopo il primo.

```
Line #1  := {
    Point (10,10) Point (50,50)
    Direction(0)
};
external Event #1  := {
    CrossedLine #1 before (0,10) CrossedLine #1
    where first.oid != second.oid
};
```

4.9.4 ObjectState

Un ObjectState è definito nel modo seguente:

```
[external]  ObjectState #<n>  := <condition>;
```

In questo caso <condition> è una variabile metasinttatica per un'espressione Booleana. L'utente può specificare fino a 32 stati (da 1 a 32) tramite il numero <n>, ma solo i primi 16 stati (<n> da 1 a 16) possono essere external. Lo stesso ObjectState non può essere definito due volte.
Diversamente da un SimpleState, che è inizializzato solo una volta, un ObjectState è inizializzato per ogni oggetto visibile nel frame elaborato. Per distinguere tra tutte le proposizioni, l'ID dell'oggetto è associato a ciascuna proposizione come un attributo. Di conseguenza la lista di attributi di un ObjectSate è:

```
ObjectState #<n> (OID:oid)  -> BOOLEAN
```

Nella proposizione <condition>, l'ID dell'oggetto può essere utilizzato per accedere alle proprietà dell'oggetto o ad un ObjectState definito precedentemente dello stesso oggetto, in aggiunta a quelli che sono permessi all'interno della proposizione <condition> di un SimpleState. La lista seguente cataloga gli stati Booleani che necessitano di un ID dell'oggetto come attributo e che possono quindi essere utilizzati come condizione di un ObjectState:

InsideField #<n> dove n è il numero di una primitiva Field

IsLoitering #<n> dove n è il numero di una primitiva Loitering

Anche le seguenti proprietà degli oggetti e stati non-Booleani sono disponibili all'interno della definizione di ObjectState:

ObjectsInField #<n> dove n è il numero di una primitiva Field

Direction

Velocity

AspectRatio

ObjectSize

Esempio

Il seguente script di attività IVA rileva oggetti che stanno effettuando un eccesso di velocità all'interno del campo specificato.

```
Field #1  := {…};
external ObjectState #1  :=
    Velocity within (30,*) and InsideField #1;
```

4.10 Eventi definiti dagli utenti

Nelle sezioni precedenti sono stati introdotti numeroso eventi che sono generati automaticamente assieme alle corrispondenti primitive. L'utente può definire nuovi eventi utilizzando relazioni temporali tra gli stessi. Oltre a relazioni temporali, possono essere formulate condizioni addizionali per limitare gli eventi in modo ulteriore. Con queste condizioni l'utente ha accesso agli attributi degli eventi e può formulare limiti su di essi. Nelle sezioni seguenti sono descritte le differenti possibilità in maggiore dettaglio.

4.10.1 Sintassi di base

Gli eventi ad hoc sono definiti nel modo seguente:

```
[external]  Event #<n>  := <event>;
```

In questo esempio <event> è la variabile metasintattica per qualsiasi evento predefinito (come EnteredField, introdotto nelle sezioni precedenti o un evento definito dall'utente) o espressioni più complesse di un evento come descritte nelle sezioni seguenti. L'utente può specificare fino a 32 eventi (da 1 a 32) tramite il numero <n>, ma solo i primi 16 eventi (<n> da 1 a 16) possono essere external. Lo stesso evento ad hoc dell'utente non può essere definito due volte. Event #<n> eredita la lista degli attributi da <event>, ossia fornisce la stessa lista di attributi di <event>.
Quando l'Alarm Task engine rileva un evento esterno definito dall'utente, la segnalazione di allarme corrispondente è stabilita per esattamente un frame elaborato. Se la stessa segnalazione di allarme è utilizzata da molteplici regole esterne (per esempio ObjectState esterno o SimpleState esterno) la segnalazione di allarme è stabilita se una delle regole è attive.
La seguente tabella riassume gli eventi predefiniti che possono essere utilizzati come variabile metasintattica <event>:

FollowedRoute #<n> dove n è il numero di una primitiva Route

CrossedLine #<n> dove n è il numero di una primitiva Line

EnteredField #<n> dove n è il numero di una primitiva Field

LeftField #<n> dove n è il numero di una primitiva Field

Appeared

Disappeared

Idled

Removed

4.10.2 Relazioni temporali

Le operazioni più interessanti sugli eventi sono le relazioni temporali. La parola chiave before combina due eventi nel modo seguente. Se il secondo

evento è azionato e il primo evento è stato azionato precedentemente, un altro evento con gli stessi attributi del secondo è fatto scattare. Nella forma più semplice la sintassi è la seguente:

```
<event>  := { <event> before <event>  }
```

Le parentesi graffe sono necessarie per evitare associazioni ambigue di costruzioni contenute una dentro l'altra.
Con l'estensione seguente si può specificare un intervallo di tempo, che limita ulteriormente la cronologia di due eventi.

```
<event>  := { <event> before (<from>,<to>) <event> }
```

In questa formulazione il primo evento è solo un candidato per il secondo evento se è accaduto almeno `<from>` secondi e al massimo `<to>` secondi prima del secondo evento. Sostituendo `<to>` con il simbolo * (asterisco), può essere definito un intervallo di tempo a partire da `<from>`.
Aggiungendo la parola chiave `not` , è inoltre possibile controllare se il primo evento non sia accaduto prima del secondo evento nell'intervallo di tempo specificato (l'intervallo di tempo è ancora una volta opzionale):

```
<event>  := { <event> not before (<from>,<to>) <event> }
```

La parola chiave `or` può essere utilizzata per azionare un evento se o l'evento A o l'evento B sia accaduto. La lista di attributi per entrambi gli eventi deve essere la medesima:

```
<event>  := <event> or <event>
```

In questo caso le parentesi graffe possono essere omesse.

Esempio

`Event` #1 segnala gli oggetti che attraversano una delle linee. `Event` #2 rileva un paio di oggetti, dove un oggetto ha attraversato la prima `Line` #1 e un oggetto ha attraversato la seconda `Line` #2 dopo al massimo cinque secondi. `Event` #3 è azionato da oggetti che attraversano la `Line` #2 mentre la `Line` #1 non era attraversata per cinque secondi.

```
Line #1  := {…}
Line #2  := {…}
external Event #1  := CrossedLine #1 or CrossedLine #2;
external Event #2  := {
    CrossedLine #1 before (0,5) CrossedLine #2
};
external Event #3  := {
    CrossedLine #1 not before (0,5) CrossedLine #2
};
```

4.10.3 Condizioni

Spesso è necessario limitare gli eventi con i loro attributi e le proprietà degli oggetti coinvolti. Nel linguaggio script delle attività IVA, si può fare questo tramite la proposizione where.

```
<event>  := { <event> where <condition> }
```

L'Alarm Task engine si azionerà solo se la <condition> è soddisfatta nel momento in cui <event> è accaduto. Quali stati e proprietà possano essere utilizzati nella proposizione <condition> dipende dalla lista di attributi di <event>. In linea di principio gil eventi con una lista di attributi vuota possono avere una proposizione <condition> simile a quella di un SimpleState. Se la lista di attributi contiene OID:oid come attributo, la proposizione <condition> è simile a quella di un ObjectState. Con i concetti introdotti finora è possibile rilevare se lo stesso oggetto ha attraversato per prima cosa una linea e successivamente un'altra. Il linguaggio script delle attività IVA risolve questa funzione con una combinazione delle parole chiave before e where.

```
<event>  := { <event> before <event> where <condition> }
```

Nella proposizione <condition> dopo la parola chiave where è possibile accedere agli attributi di entrambi gli eventi coinvolti nella relazione before. Un attributo <x> dell'evento a sinistra può essere accessibile tramite first.<x>, mentre gli attributi dell'evento a destra sono disponibili tramite second.<x>. Se un attributo <x> appartiene esclusivamente ad uno dei due eventi, la specificazione di first e second è opzionale. Se un attributo <x> appartiene ad entrambi gli eventi, l'attributo è associato all'evento second, che è il più recente. Qualsiasi delle altre varianti di before descritte nel paragrafo "Relazioni temporali" possono essere combinate in modo simile con una proposizione <condition>.

Esempio

Con queste estensioni possono essere trovati gli oggetti che attraversano due linee consecutivamente ad una data velocità, come mostrato da Event #1.

```
Line #1  := {…}
Line #1  := {…}
external Event #1  := {
    CrossedLine #1 before (0,5) CrossedLine #2
    where first.oid == second.oid and
       Velocity (second.oid) within (30,*)
};
```

4.10.4 Cambiamenti di stato

Per rilevare per esempio cambiamenti nell'aspetto di un oggetto, sono introdotte le parole chiave `OnChange`, `OnSet` e `OnClear`. Possono essere combinate con uno stato definito dall'utente e azionare un evento se lo stato modifica il suo valore, se lo stato diventa `true`, o se lo stato diventa `false`. La sintassi è la seguente:

```
<event>  := OnChange <state>
<event>  := OnSet <state>
<event>  := OnClear <state>
```

Nell'esempio `<state>` è un `ObjectSate #<n>` definito precedentemente o un `SimpleState #<n>`. La lista degli attributi dello `<state>` è passata al corrispondente evento di cambiamento. Per esempio, la lista degli attributi di un evento `OnChange SimpleState #<n>` è vuota, mentre la lista di attributi di un `OnSet ObjectState #<n>` ha `OID:oid` come suo unico attributo.

Esempio

Il seguente script di attività IVA rileva gli oggetti che stanno modificando la loro forma da alto e stretto a piatto e largo.

```
ObjectState #1  := AspectRatio within (1.2,*);
ObjectState #2  := AspectRatio within (*, 0.8);
external Event #1  := {
    OnClear ObjectState #1 before OnSet ObjectState #2
    where first.oid == second.oid
};
```

`ObjectState #1` è `true` se l'oggetto è più alto che largo. `ObjectState #2` è `true` se l'oggetto è più largo che alto. Per gli oggetti che sono quasi quadrati entrambi gli stati sono `false`. L'attività cerca oggetti il cui `ObjectState #1` è stato deciso un po' di tempo prima e il cui `ObjectState #2` è deciso ora. Pertanto, è sufficiente aspettare per il `OnSet ObjectState #2` e controllare se `OnClear ObjectState #1` è accaduto precedentemente.