POLITECNICO DI MILANO
V Facoltà di Ingegneria
Corso di Laurea Triennale in Ingegneria Informatica
Anno Accademico 2009/2010

LINGUAGGIO PERLA: SINTASSI PER LA CREAZIONE DI UN MODELLO DI CONTESTO

Relatori:
Chiar.mo prof. Letizia TANCA
Chiar.mo prof. Fabio SCHREIBER

Studenti:
Filippo GIOVE matr.713890
Davide LONGONI matr.712850

INDICE

INTRODUZIONE

Una nuova generazione di applicazioni informatiche chiamate “Pervasive System Context-aware” sta nascendo.

Queste applicazioni sono caratterizzate dalla possibilità di riconoscere particolari situazioni, in modo da elaborare e filtrare le informazioni richieste dall’utente in modo diverso a seconda del contesto in cui l’utente viene a trovarsi.

Spesso la capacità di discretizzare il contesto dell’applicativo è permessa dall’uso intensivo di sensori, collegati tra loro per formare una vera e propria rete ad-hoc (Wireless Sensors Network).

Per determinare il contesto, le applicazioni devono essere in grado di interpretare i dati derivanti da queste WSN.

In quest’ambito si colloca il linguaggio PerLa, il quale potrà ritagliarsi un ruolo molto importante per l’implementazione di queste applicazioni.

Introduzione a Perla

Perla (PERvasive LAnguage) è un linguaggio completamente dichiarativo che permette all’utente di interrogare un sistema pervasivo in modo simile a come interrogherebbe una base di dati utilizzando SQL.

Per linguaggio dichiarativo si intende un linguaggio con cui si stabilisce che cosa debba fare il programma, cioè che relazione intercorre fra input e output, a differenza dei linguaggi imperativi, con i quali si stabilisce come un problema deve essere risolto, cioè come derivare un output da uno o più input.

Un sistema pervasivo è una grande rete eterogenea composta da diversi dispositivi, ognuno dei quali può utilizzare diverse tecnologie, come ad esempio reti di sensori wireless (WSN), sistemi RFID, GPS e molti altri tipi di sensori.

Lo scopo principale del linguaggio PerLa è quindi nascondere all’utente l’elevata complessità della programmazione di basso livello e le elevate dimensioni del sistema, dovuta all’eterogeneità e al numero dei sensori con cui si andrà ad operare, fornendogli un’interfaccia simile a quella di una base di dati, in modo che possa recuperare i dati dal sistema con semplicità e velocità.

Scopo del progetto

Lo scopo di questo progetto è di estendere il linguaggio PerLa, introducendo una sintassi che permetta di rendere il linguaggio context-aware, ossia sensibile al contesto.
Grazie al linguaggio si potrà, tramite rilevazioni ed elaborazioni di dati, contestualizzare la situazione dell'utente, con benefici sia per quest'ultimo, che ridurrà la sua interazione con il sistema ricevendo da esso le sole informazioni utili in quella determinata situazione (riduzione rumore informativo), sia per la WSN poiché implicitamente si introdurrà una politica di risparmio energetico delle batterie dei sensori, componente spesso critica per questi dispositivi.
La presentazione del progetto inizierà con una breve digressione sul significato del concetto di contesto, presentando alcune definizioni presenti in letteratura.
Successivamente si presenterà il modello scelto per la rappresentazione del contesto, e le relative sintassi per la sua implementazione in Perla.
Si utilizzerà come caso di studio il progetto dell'ART DECO, riguardante l'automazione dei sistemi di controllo della qualità della produzione vinicola, dalla vigna fino alla tavola.
Infine si tratterà la possibilità di introdurre "politiche di scheduling" per gestire situazioni in cui più contesti sono attivabili contemporaneamente.

Il contesto in un sistema pervasivo

Oggigiorno gli utenti dei sistemi informativi possono venire a contatto con una grande quantità di dati, non sempre tutti utili al processo che stanno svolgendo, trovandosi immersi nel cosiddetto "rumore informativo".
Nell'ambito delle WSN ridurre il rumore informativo comporta un ulteriore beneficio poiché riducendo le rilevazioni alle sole informazioni strettamente necessarie, si ottiene un sostanziale risparmio di energia: quindi l'introduzione di una struttura appropriata che possa gestire cambiamenti di contesto a run-time comporta diversi benefici, sia dal punto di vista dell'utente che da quello applicativo.
Urge allora la necessità di adattare i dati alle richieste delle singole applicazioni o dei singoli utenti, effettuando il cosiddetto "Data Tailoring", cioè ritagliare su misura le informazioni da presentare a fronte di una richiesta specifica. Le "forbici" utilizzate per questo processo possono essere rappresentate dal contesto.
La parola contesto deriva dal latino *cum* (con o assieme) e *texere* (tessere), descrive il contesto non

solo come un profilo, ma anche come un processo *attivo relativo a come gli uomini utilizzano la loro esperienza all'interno del proprio ambiente per fornirle un significato.*

Innanzitutto è utile determinare verso cosa, o chi il contesto è orientato.

Molti applicativi utilizzano il concetto di contesto per adattare in modo diverso la presentazione di dati in base ai diversi canali informativi o dispositivi utilizzati (**Orientati alla presentazione)**.

Altri invece attraverso coordinate locali e temporali modellano le informazioni in maniera differente con grande flessibilità e accortezza (**Orientati alla località**).

Altri ancora hanno la capacità di osservare ed elaborare le attività dell'utente per creare uno "storico" che sarà successivamente utile per inferire le attività future dell'utente (**Orientato sull'utente**).

In letteratura sono presenti tantissime definizioni di contesto, qui di seguito è riportata la più significativa per il nostro ambito di studio.

Il contesto viene definito come qualsiasi informazione che può essere usata per caratterizzare, rappresentare la situazione di un'entità.

Un'entità è una persona, un luogo o un oggetto che è considerato rilevante per l'interazione tra utente e applicazione (includendo l'utente e l'applicazione stessa).[1]

Questa interazione è rappresentata da molteplici variabili, i cui cambiamenti devono essere rilevati per poter adattare di conseguenza la comunicazione tra utente e applicativo.

PerLa, al momento, consente solo un livello base di attuazione di query contestuali attraverso l'operatore PILOT JOIN, qui di seguito brevemente descritto.

PILOT JOIN

L'obiettivo principale della PILOT JOIN consiste nell'aumentare le capacità del linguaggio permettendo l'esecuzione di semplici query contestuali, che permettono anche un sostanziale risparmio energetico e di tempo di calcolo.
La sintassi della funzione è rappresentata dalla seguente struttura:

<Pilot Join Clause> PILOT JOIN <Correlated Table List>
*<Correlated Table List> <Correlated Table> { ',' <Correlated Table> }**
<Correlated Table> <Data Structure Name> ON <Condition>

Per spiegare la sintassi consideriamo un esempio tratto da [2].
Si suppone di voler monitorare alcuni container caricati su una nave. Ogni container possiede due sensori, uno di luminosità situato all'esterno e uno di temperatura situato all'interno del container. Se per esempio volessimo monitorare la temperatura dei container che sono direttamente esposti alla luce solare, occorrerebbe un costrutto particolare che possa permettere di rilevare solo la temperatura dei container esposti alla luce solare (rilevabile con il primo tipo di sensore), ignorando completamente gli altri. Questa query è possibile effettuarla in Perla grazie appunto alla clausola PILOT JOIN.

Prima di tutto bisogna creare una query che monitori la luminosità di tutti i container, per far ciò utilizzeremo uno SNAPSHOT:

```
CREATE SNAPSHOT UnderTheSun (sensorID ID, light FLOAT, containerID ID)
WITH DURATION 1 h AS
LOW:
SELECT ID, light, containerID
SAMPLING EVERY 1 h
WHERE light > [threshold]
EXECUTE IF deviceType = "LightSensor"
```

Questa query sarà effettuata solamente su tutti i sensori che rilevano una luminosità maggiore rispetto ad una certa soglia (threshold).

Il passo successivo è quello di selezionare le temperature dei sensori appartenenti ai container esposti alla luce solare. Per rispettare questo requisito dobbiamo usare la funzione di PILOT JOIN come segue:

```
CREATE OUTPUT STREAM Temperatures(sensorID ID, temp FLOAT, containerID ID) AS
LOW:
EVERY ONE
SELECT ID, temp, containerID
SAMPLING EVERY 1 m
PILOT JOIN UnderTheSun ON UnderTheSun.containerID = containerID
EXECUTE IF EXISTS (ALL)
```

In questa query la funzione PILOT JOIN permette di creare un collegamento logico tra il sensore di luminosità e il sensore di temperatura dei container, selezionando solo i sensori di temperatura con ID del container presente nello SNAPSHOT UnderTheSun, ossia quei container che superano una certa soglia di luminosità.
La funzione di PILOT JOIN, intesa come funzione contestuale, per PerLa presenta alcune limitazioni:

- Il collegamento tra due classi di sensori deve essere esplicito.
  Come visto nell'esempio precedente per creare la correlazione tra i dati si è dovuta inserire una condizione esplicita che permetteva tale collegamento. Nell'esempio si è trovata una condizione basata su dei dati, ma se la condizione dipende da collegamenti i cui valori non sono presenti in alcuna struttura dati, allora la correlazione tramite PILOT JOIN è impossibile.

- La funzione PILOT JOIN non è abbastanza potente per query contestuali più complesse.
  Se per correlare dei dati ho bisogno di verificare più di una condizione, bisogna implementare una query che utilizza delle PILOT JOIN in cascata aumentando la complessità di implementazione e di comprensione.

- La funzione non permette lo scambio di informazioni con entità esterne.
  Le WSNs vengono molto spesso utilizzate per monitorare fenomeni di elevate dimensioni che non solo dipendono da dati rilevabili dai sensori ma anche da fattori esterni.

- Non è possibile gestire il tempo.
  Le strutture SNAPSHOT e STREAM non permettono di gestire parametri sensibili al tempo e di conseguenze neanche il PILOT JOIN che le utilizza.

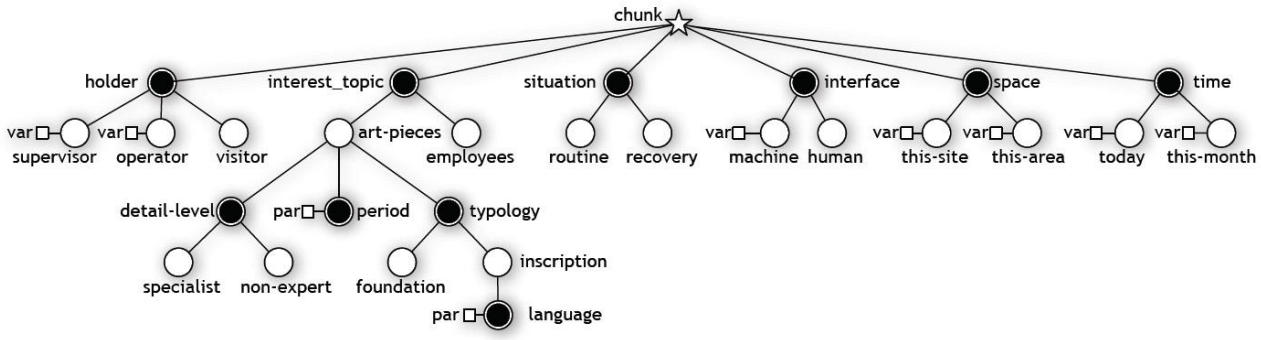
Una possibile soluzione a queste limitazioni è l'introduzione di una struttura dati che permetta la gestione esplicita del contesto: il Context Dimension Tree.

MODELLO DI CONTESTO

Il Context Dimesion Tree

In questo progetto si modellizzerà il contesto attraverso l'uso del Context Dimension Tree introdotto in [3].
Principalmente si rappresenta il contesto come la congiunzione di prospettive, con i relativi valori, dei vari aspetti del contesto in cui l'utente si colloca.
Il modello di contesto, chiamato Context Dimension Tree (CDT), è stato definito con lo scopo di filtrare i dati relativi a una particolare applicazione, in base alle reali necessità dell'utente.
Esso è una rappresentazione ad albero dell'insieme delle possibili prospettive e dei relativi valori, utili alla descrizione dei diversi aspetti del contesto nel quale un utente si colloca.

Esempio di CDT relativo ad un sito archeologico [4]



In questa rappresentazione i nodi neri rappresentano le cosiddette dimensioni, cioè le varie prospettive costituenti di un contesto. I nodi bianchi rappresentano invece tutti i possibili valori di queste dimensioni e sono chiamati nodi-concetto.
Nella rappresentazione gerarchica dei contesti, così definita, ogni nodo specifica in modo più dettagliato gli aspetti descritti dai suoi antenati.
Alcune dimensioni (o sottodimensioni) non hanno figli che sono nodi-concetto, ma hanno semplicemente un parametro. Ciò succede quando una dimensione ha un eccessivo numero di possibili valori, per questo si è deciso di introdurre un nodo parametro che sarà istanziato al momento dell'esecuzione con il reale valore.
I parametri possono essere aggiunti anche a nodi-concetto (nodi bianchi) quando si desidera che l'istanza del relativo concetto venga filtrata al momento dell'esecuzione in base ad un opportuno

valore.
Il nostro lavoro verterà all'introduzione di una sintassi che possa permettere di costruire tale albero, le cui dimensioni saranno utilizzate successivamente per la definizione della sintassi del contesto.

Sintassi per la creazione delle dimensioni

Si introduce qui di seguito la sintassi per la creazione del CDT (Context Dimension Tree), le cui dimensioni saranno parte fondamentale della sintassi di creazione di un contesto.
La sintassi permette la creazione a "pezzi" del CDT, ad ogni nuova dimensione definita, un nuovo ramo (dimensione) con le relative foglie (valori delle dimensioni) viene aggiunto all'albero.
Innanzitutto, come descritto in [5], le dimensioni con i relativi nodi-concetto possono essere di tre tipi:

- dimensioni con nodi-concetto standard;
- dimensioni con parametro;
- dimensioni con nodo-concetto con parametro.

Per questo motivo ci saranno tre diverse sintassi, simili ma con parti differenti, proprio per poter rappresentare queste tre tipologie.

Sintassi dimensioni con nodi-concetto standard o nodo-concetto con parametro

CREATE DIMENSION *<Namedimension>*
SPECIFY *<FatherNode's Name>*
{
<value>$^{+}$
}

<value> → CREATE VALUE *<Namevalue>* | *<Namevalue>* (*<ParameterList>*$^{+}$)
[VALID WHEN *<BooleanExpression>*]
[RELATIVE SENSORS: *<Query>*]

Sintassi dimensioni con parametro

CREATE DIMENSION *<Namedimension>*
SPECIFY *<FatherNode's Name>*
{
WITH PARAMETER *<NameParameter>*
[VALID WHEN *<BooleanExpression>*]
[RELATIVE SENSORS: *<Query>*]
}

In *Namedimension* si definisce il nome della dimensione.

Attraverso il costrutto SPECIFY si indica il nome del nodo-valore padre di cui la dimensione è figlia diretta. Per le dimensioni di primo livello il nodo-valore sarà ovviamente il nodo ROOT, che rappresenta la prospettiva più generale sulla WSN.

Nella clausola VALID WHEN è presente una condizione booleana che descrive le condizioni per le quali la dimensione è rappresentata da quel determinato valore. Ogni nodo-concetto eredita, dal nodo-concetto padre della dimensione cui si riferisce, tutte le condizioni della VALID WHEN. Se per un nodo non viene specificata risulta essere equivalente a quella del padre.

Il costrutto RELATIVE SENSORS attraverso un'apposita query Perla estrae tutti gli identificatori dei sensori che raccolgono informazioni utili, pertinenti al nodo-concetto. La clausola è opzionale, poiché il nodo-concetto può dipendere da fattori esterni alla WSN, quindi si può non specificare oppure scrivere RELATIVE SENSORS: ALL (le due soluzioni sono semanticamente equivalenti).

Per esempio se volessimo definire un nodo-concetto chiamato "in ufficio", nella query contenuta in RELATIVE SENSORS verranno estratti tutti i sensori che risultano, attraverso opportune rilevazioni GPS, all'interno dell'ufficio. L'utilità di questa selezione sarà più comprensibile quando si introdurrà la sintassi di creazione del contesto.

Inoltre la query contenuta nella clausola RELATIVE SENSORS, del nodo figlio, verrà effettuata solo sui sensori selezionati dal nodo padre.

Esempio

```
CREATE DIMENSION Namedimension
SPECIFY  Root
{   CREATE VALUE PadreValue
     VALID WHEN PadreBooleanExpression
     RELATIVE SENSORS query                    //dalla query vengono selezionati i sensori
…… }                                             ID1  ID2  ID3

CREATE DIMENSION Namedimension
SPECIFY  PadreValue
{   CREATE VALUE FiglioValue
    VALID WHEN FiglioBooleanExpression     //in realtà la valid when è formata da
                                             FiglioBooleanExpr AND PadreBooleanExpr

   RELATIVE SENSORS query   }          //la query verrà effettuata su ID1 ID2 ID3 e in uscita
                                         risulterà un sottoinsieme dei tre sensori selezionati
```

Scrivendo ulteriori condizioni nei nodi figli si va a specificare più in dettaglio la prospettiva generata dal nodo padre.

Analizzando la sintassi della creazione delle dimensioni con nodo-concetto standard, all'interno delle parentesi graffe, vengono elencati e definiti tutti i nodi-valori della dimensione. Per ciascuno

di essi si definisce il nome del valore (*Namevalue*), successivamente attraverso la clausola VALID WHEN si definisce un insieme di condizioni booleane che identifica tutte le condizioni utili per definire, correttamente e in modo esaustivo, il nodo-valore.
Con riferimento alla sintassi della creazione delle dimensioni con parametro, all'interno delle parentesi graffe, viene definito il parametro che permetterà di discretizzare a run-time la dimensione evitando la creazione esaustiva di tutti i valori possibili.
Questa soluzione è utilizzata quando il numero dei diversi valori per tali dimensioni risulterebbe troppo elevato.
Sia la VALID WHEN sia la clausola RELATIVE SENSORS posso essere parametrizzate dal valore che assume il parametro.

Esempio

```
CREATE DIMENSION Temp°C
SPECIFY ….
{
        WITH PARAMETER  $over_temp
        VALID WHEN EXISTS (CREATE OUTPUT STREAM $tempSensors(nodeId ID) AS
                                LOW:
                                        EVERY ONE
                                        SELECT ID
                                        SAMPLING
                                                EVERY 1h
                                                WHERE temperature > $over_temp
                                EXCUTE IF EXIST (temperatura))
        RELATIVE SENSORS: $tempSensors(nodeId ID)
}
```

In ultima analisi, prendiamo in considerazione la sintassi delle dimensioni con nodo-concetto con parametro. In questo caso il valore del nodo-concetto è definito dal valore di un o più parametri.
La sintassi è simile a quella del caso standard, ma si differenzia dal fatto che i valori della dimensione dipendono da uno o più parametri.

Esempio

```
CREATE DIMENSION Time
SPECIFY Root
{
        CREATE VALUE Periodo($inizio, $fine)
        VALID WHEN TIMESTAMP IS BETWEEN $inizio AND $fine

}
```

Sintassi per la creazione del contesto

Secondo il modello utilizzato, un contesto è definito come la congiunzione di proposizioni del tipo $dim_1 = val_1$, dove ogni coppia (dim, value) rappresenta la dimensione ed il relativo valore che sono utili nella discretizzazione del contesto.
La sintassi proposta per la creazione del contesto è la seguente:

CREATE CONTEXT <*Namecontext*>
ACTIVE IF <*DimList*>
[ON ENABLE <*contextual query*>]
[ON DISABLE <*actualization query*>]
REFRESH <*condition*>

<*DimList*> → <*Namedimension*> = <*value*> { AND <*Namedimension*> = <*value*> }*
<*condition*> → EVERY <*duration*> | ON EVENT <*eventname*>

Il costrutto ha nella clausola ACTIVE IF la sua componente più importante. In essa vengono elencate la lista dei valori delle sole dimensioni che sono utili per rappresentare il contesto in creazione.

Esempio

ACTIVE IF Dim1=Value1_of_Dim1 AND Dim3=Value5_of_Dim3 AND
Dim5=Value2_of_Dim5....

Per ogni dimensione presente deve essere verificata la veridicità della VALID WHEN del relativo valore.
Il contesto sarà attivabile se e solo se tutte le VALID WHEN di tutti i valori di dimensioni presenti nella ACTIVE IF sono vere.
Nella ON ENABLE è presente la query contestuale da effettuare nel momento in cui il contesto viene attivato.
Tale query verrà effettuata solo sui sensori che fanno parte di tutte le "liste" di sensori selezionate dai RELATIVE SENSORS di tutti i valori di dimensioni presenti nella ACTIVE IF .

Esempio

Sensor of Value1_of_Dim1
ID3
ID4
ID5
ID7

Sensor of Value1_of_Dim3
ID3
ID5
ID12

Sensor of Value1_of_Dim5
ID1
ID2
ID3
ID5
ID15

CREATE CONTEXT *Namecontext*

ACTIVE IF Dim1=Value1_of_Dim1 AND Dim3=Value5_of_Dim3 AND

Dim5=Value2_of_Dim5

ON ENABLE {contextual *query*}$^{+}$

ON DISABLE {*actualization query*}$^{+}$

REFRESH *time*

La query contestuale verrà effettuata solo sui sensori ID3 e ID5.

In ON DISABLE vengono effettuate le operazioni necessarie quando la ACTIVE IF non risulta più vera.

Per esempio si possono disabilitare le query contestuali utilizzando la funzione DROP seguito dal nome, oppure da un elenco di query che si desidera eliminare una volta disattivato il contesto.

La clausola REFRESH indica sotto quali condizioni avviene la rivalutazione dell'ACTIVE IF.

Ciò può avvenire mediante due modalità: ad ogni intervallo temporale o al verificarsi di un evento .

Esempio

REFRESH EVERY 1h

REFRESH ON EVENT eventoX

Nel primo caso si rivaluta la ACTIVE IF ogni ora, nel secondo caso solo quando si presenta l'evento "eventoX ".

IL CASO DI STUDIO

Questo riportato di seguito è uno dei casi di studio dell'ART DECO [6], progetto istituito dal MIUR (Ministero dell'Istruzione, dell'Università e della Ricerca), in particolare questo caso di studio riguarda la produzione vinicola.

Alcuni parametri del ciclo di produzione devono essere tenuti sotto stretta osservazione per garantire la qualità finale del prodotto e per permettere una completa tracciabilità: dalla fase di coltivazione dell'uva, fino alla fase di etichettatura delle bottiglie.

La prima fase di rilevamento è effettuata attraverso sensori di umidità e di temperatura posizionati nella vigna e connessi l'uno all'altro attraverso una rete ad-hoc.

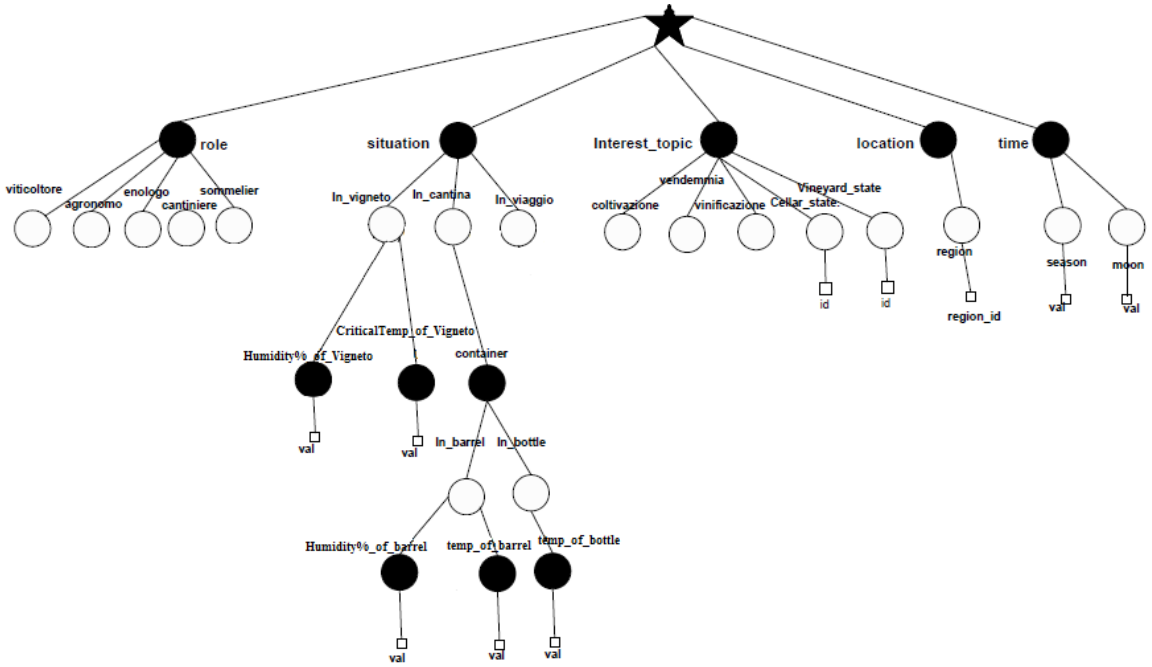
Un coltivatore equipaggiato con un PDA può rilevare i parametri ambientali circostanti (temperatura e se possibile umidità) e la sua posizione real time.

In seguito alla fase di coltivazione avviene un'altra fase critica del ciclo di produzione, il trasporto.

Occorre monitorare le temperature delle bottiglie per evitare sbalzi termici, per questo ciascun imballaggio ha un sensore di temperatura per rilevare la temperatura delle bottiglie al suo interno, oltre ad un GPS e ad una base station per controllarne la posizione.

L'ultimo tipo di rilevamento è effettuato per monitorare le bottiglie all'interno delle cantine. Tutte le bottiglie sono equipaggiate con un'etichetta RFID, mentre un lettore di RFID è montato all'ingresso di ogni cantina, così facendo è possibile determinare con sicurezza quando una bottiglia entra ed esce da esse.

Nella figura sottostante è illustrato il Context Dimension Tree.



Creazione del CDT relativo al caso di studio

Per la creazione delle dimensioni e dei suoi nodi-concetto si utilizzano alcune funzioni di supporto. Ad esempio isViticoltore($ID) e simili sono funzioni che, dato l'identificativo personale di chi utilizza l'applicativo, verificano se esso fa parte o meno di una categoria di lavoratori del vigneto.
Si presuppone quindi la presenza di tante tabelle statiche quante sono le tipologie di lavoratori, nelle quali sono elencati tutti gli ID dei componenti di tale categoria.
Simili sono le funzioni is_in_Cellar(locationX, locationY) e is_in_Vineyard(locationX, locationY). Esse hanno come parametri delle coordinate spaziali e verificano se sono incluse rispettivamente nella vigna o nella cantina.
Inoltre, si è scelto di utilizzare come struttura dati degli SNAPHOT, poiché sono più adatti per modellare la lista dei sensori che sono utili per un dato nodo-concetto. Infatti, essi hanno una durata (determinata da WITH DURATION). Una volta terminato tale periodo, tutti i componenti dello SNAPHOT vengono cancellati.

RFID TAG			
Nome	Tipo del dato	categoria	Descrizione
ID	ID	ID	identificatore dell'oggetto logico
lastReaderID	ID	NP	ID dell'ultimo reader che ha rilevato il tag
lastReaderChanged	-	E	Notifica che il tag è stata rilevata da un reader
deviceType	STRING	S	Tipologia dispositivo

Tabella 1

WSN NODE			
Nome	Tipo del dato	categoria	Descrizione
ID	ID	ID	identificatore dell'oggetto logico
temperature	FLOAT	P	temperatura rilevata
humidity	FLOAT	P	umidità rilevata
locationX	FLOAT	S	coordinate del sensore, coordinate X
locationy	FLOAT	S	coordinate del sensore, coordinate Y
deviceType	STRING	S	Tipologia dispositivo
powerLevel	FLOAT	P	livello attuale di alimentazione del sensore

Tabella 2

Nella tabelle sono elencati i vari attributi che costituiscono l'oggetto logico che modellizza un RFID TAG (Tabella 1) e un nodo generale della WSN (Tabella 2).

- CREAZIONE DI DIMENSIONI DI PRIMO LIVELLO

CREATE DIMENSION Role
SPECIFY Root
{
CREATE VALUE Viticoltore
VALID WHEN isViticoltore($ID_EMPLOYED) AND is_in_Vineyard(locationX, locationY) AND TIMESTAMP BETWEEN 6.00 a.m AND 10.00 a.m OR 5.00 p.m AND 9.00 p.m
RELATIVE SENSORS:
CREATE OUTPUT SNAPSHOT ViticoltoreSensors (nodeId ID)
WITH DURATION 1h AS
LOW:
SELECT ID
SAMPLING
EVERY 1h
EXCUTE IF is_in_Vineyard(locationX,locationY)

CREATE VALUE Agronomo
VALID WHEN isAgronomo($ID_EMPLOYED) AND TIMESTAMP BETWEEN 10.00 a.m AND 5.00 p.m.
RELATIVE SENSORS:
CREATE OUTPUT SNAPSHOT AgronomoSensors (nodeId ID)
WITH DURATION 1h AS
LOW:
EVERY ONE
SELECT ID
SAMPLING
EVERY 1h

CREATE VALUE Enologo
VALID WHEN isEnologo($ID_EMPLOYED) AND TIMESTAMP BETWEEN 10.00 a.m AND 5 p.m.
RELATIVE SENSORS:
CREATE OUTPUT SNAPSHOT EnologoSensors (nodeId ID)
WITH DURATION 1h AS
LOW:
EVERY ONE
SELECT ID
SAMPLING
EVERY 1h
EXCUTE IF is_in_Cellar(locationX,locationY) OR is_in_Vineyard(locationX,locationY)

CREATE VALUE Cantiniere
VALID WHEN isCantiniere($ID_EMPLOYED) AND TIMESTAMP BETWEEN 10.00 a.m AND 5.00 p.m.
RELATIVE SENSORS:
CREATE OUTPUT SNAPSHOT CantiniereSensors (nodeId ID)
WITH DURATION 1h AS
LOW:
EVERY ONE
SELECT ID
SAMPLING
EVERY 1h
EXCUTE IF is_in_Cellar(locationX,locationY)

```
        CREATE VALUE Sommelier
        VALID WHEN isSommelier($ID_EMPLOYED) AND TIMESTAMP BETWEEN 10.00 a.m AND 5.00 p.m.
        RELATIVE SENSORS:
                CREATE OUTPUT SNAPSHOT SommelierSensors (nodeId ID)
                WITH DURATION 1h AS
                LOW:
                        EVERY ONE
                        SELECT ID
                        SAMPLING
                                EVERY 1h
                        EXCUTE IF is_in_Cellar(locationX,locationY)

}


CREATE DIMENSION Situation
SPECIFY  Root
{
        CREATE VALUE In_Vigneto
        VALID WHEN is_in_Vineyard(locationX, locationY)
        RELATIVE SENSORS:
                CREATE OUTPUT SNAPSHOT In_VignetoSensors (nodeId ID)
                WITH DURATION 1h AS
                LOW:
                        EVERY ONE
                        SELECT ID
                        SAMPLING
                                EVERY 1h
                        EXCUTE IF is_in_Vineyard(locationX,locationY)

        CREATE VALUE In_Cantina
        VALID WHEN  is_in_Cellar(locationX, locationY)
        RELATIVE SENSORS:
                CREATE OUTPUT SNAPSHOT In_CantinaSensors (nodeId ID)
                WITH DURATION 1h AS
                LOW:
                        EVERY ONE
                        SELECT ID
                        SAMPLING
                                EVERY 1h
                        EXCUTE IF is_in_Cellar(locationX,locationY)

        CREATE VALUE In_Viaggio
        VALID WHEN isCamionista ($ID_EMPLOYED)
        RELATIVE SENSORS:
                CREATE OUTPUT SNAPSHOT In_ViaggioSensors (nodeId ID)
                WITH DURATION 1h AS
                LOW:
                        EVERY ONE
                        SELECT ID
                        SAMPLING
```

```
                    ON EVENT LastReaderChanged
                    WHERE lastReaderId= Truck_of($ID_EMPLOYED)
}
```

Si presuppone che le informazioni rilevabili durante il trasporto siano utili solo ai camionisti, che dovranno tenere sempre sotto osservazione le bottiglie che trasportano e registrarne le condizioni.
Inoltre in questo nodo-valore viene introdotta una nuova funzione (Truck_of ($id)) utilizzata per identificare l'ID del lettore montato sul camion guidato da un determinato camionista ($ID_EMPLOYED).

```
CREATE DIMENSION Interest_Topic
SPECIFY  Root
{
        CREATE VALUE Coltivazione
        VALID WHEN (Role = viticoltore OR Role = agronomo) AND NOT Time = Season(21/12, 21/03)
        RELATIVE SENSORS:
                CREATE OUTPUT SNAPSHOT ColtivazioneSensors ( nodeId ID)
                WITH DURATION 1h AS
                LOW:
                        EVERY ONE
                        SELECT ID
                        SAMPLING
                                EVERY 1h
                        EXECUTE IF is_in_Vineyard ( locationX, locationY )

        CREATE VALUE Vendemmia
        VALID WHEN DATE > 01/09 AND DATE < 30/10
        RELATIVE SENSORS:
                CREATE OUTPUT SNAPSHOT VendemmiaSensors ( nodeId ID)
                WITH DURATION 1h AS
                LOW:
                        EVERY ONE
                        SELECT ID
                        SAMPLING
                                EVERY 1h
                        EXECUTE IF is_in_Vineyard ( locationX, locationY)

        CREATE VALUE Vinificazione
        VALID WHEN DATE > 31/10 AND DATE < 15/11      //la vinificazione dura all'incirca 15 giorni
        RELATIVE SENSORS:
                CREATE OUTPUT SNAPSHOT VinificazioneSensors ( nodeId ID)
                WITH DURATION 1h AS
                LOW:
                        EVERY ONE
                        SELECT ID
                        SAMPLING
```

```
                    EVERY 1h
                EXECUTE IF is_in_Cellar (locationX,locationY)

    CREATE VALUE Stato_Cantina ($id_Cellar )
    VALID WHEN Location = In_Cantina
    RELATIVE SENSORS:
            CREATE OUTPUT SNAPSHOT Stato_CantinaSensors ( nodeId ID)
            WITH DURATION 1h AS
            LOW:
                EVERY ONE
                SELECT ID
                SAMPLING
                    EVERY 1h
                EXECUTE IF is_in_Cellar ( locationX, locationY,$id_Cellar )

    CREATE VALUE Stato_Vigna ( $id_vigneto )
    VALID WHEN Location = In_Vigneto
    RELATIVE SENSORS:
            CREATE OUTPUT SNAPSHOT Stato_VignaSensors ( nodeId ID)
            WITH DURATION 1h AS
            LOW:
                EVERY ONE
                SELECT ID
                SAMPLING
                    EVERY 1h
                EXECUTE IF is_in_Vineyard ( locationX, locationY, $id_vigneto )
}
```

Le funzioni is_in_Cellar (locationX, locationY,$id_Cellar) e is_in_Vineyard (locationX, locationY, $id_vigneto) sono un overloading delle funzioni precedentemente specificate, permettendo attraverso un terzo parametro di identificare univocamente un particolare vigneto o una particolare cantina, nel caso ce ne siano più di uno.

```
CREATE DIMENSION Location
SPECIFY  Root
{
    CREATE VALUE Region($id_region)
    VALID WHEN  is_in_Region($id_region)
 }


CREATE DIMENSION Time
SPECIFY  Root
{
    CREATE VALUE Season ( $inizio, $fine )
    VALID WHEN DATE IS BETWEEN $inizio AND $fine

    CREATE VALUE Moon ($FaseLunare)
    VALID WHEN  IS_IN_FASELUNARE($FaseLunare)
 }
```

- ESEMPIO CREAZIONE DIMENSIONE DI SECONDO LIVELLO

```
CREATE DIMENSION Humidity%_of_Vigneto
SPECIFY In_Vigneto
{
        WITH PARAMETER  $humidity
        VALID WHEN  EXISTS (
                        CREATE OUTPUT SNAPSHOT Humidity%_of_VignetoSensors(nodeId ID)
                        WITH DURATION 1h AS
                        LOW:
                                EVERY ONE
                                SELECT ID
                                SAMPLING
                                        EVERY 1h
                                        WHERE humidity = $humidity
                                EXCUTE IF EXISTS (humidity)      )
        RELATIVE SENSORS:   Humidity%_of_VignetoSensors(nodeId ID)
}


CREATE DIMENSION CriticalTemp_of_Vigneto
SPECIFY In_Vigneto
{
        WITH PARAMETER  $CriticalTemp
        VALID WHEN  EXISTS (
                        CREATE OUTPUT SNAPSHOT Critical_of_VignetoControl(nodeId ID)
                        WITH DURATION 1h AS
                        LOW:
                                EVERY ONE
                                SELECT ID
                                SAMPLING
                                        EVERY 1h
                                        WHERE temperature <  $CriticalTemp
                                EXCUTE IF EXISTS (temperature) )
        RELATIVE SENSORS: Critical_of_VignetoControl(nodeId ID)
}


CREATE DIMENSION Container
SPECIFY In_cantina
{
        CREATE VALUE In_barrel
        RELATIVE SENSORS:
                        CREATE OUTPUT SNAPSHOT In_barrelSensors (nodeId ID)
                        WITH DURATION 1h AS
                        LOW:
                                EVERY ONE
                                SELECT ID
                                SAMPLING
                                        EVERY 1h
                                EXCUTE IF Is_on_barrel(ID)
```

```
CREATE VALUE In_bottle
RELATIVE SENSORS:
        CREATE OUTPUT SNAPSHOT In_bottleSensors (nodeId ID)
        WITH DURATION 1h AS
        LOW:
            EVERY ONE
            SELECT ID
            SAMPLING
                EVERY 1h
            EXCUTE IF  Is_on_bottle(ID)

}
```

La condizione della VALID WHEN è la stessa del nodo padre (is_in_Cellar(locationX, locationY)), quindi non viene riscritta, poiché un nodo figlio eredita di default tutte le condizioni definite nel padre.

Is_on_barrel(ID) e Is_on_bottle(ID) sono funzioni che verificano che l'ID sia relativo ad un oggetto logico rappresentante qualsiasi sensore montato su un barile o su una bottiglia (si presuppone la presenza di una tabella statica contenente la lista di tali ID montanti su barili o bottiglie).

- ESEMPIO CREAZIONE DIMENSIONI DI TERZO LIVELLO

```
CREATE DIMENSION Humidity%_of_barrel
SPECIFY In_barrel
{
    WITH PARAMETER  $humBarrel
    VALID WHEN  EXISTS (
        CREATE OUTPUT SNAPSHOT Humidity%_of_barrelControl(nodeId ID)
        WITH DURATION 1h AS
        LOW:
            EVERY ONE
            SELECT ID
            SAMPLING
                EVERY 1h
                WHERE humidity= $humBarrel
            EXCUTE IF EXISTS (humidity)      )
    RELATIVE SENSORS:  Humidity%_of_barrelControl(nodeId ID) }
```

```
CREATE DIMENSION temp_of_barrel
SPECIFY In_barrel
{
    WITH PARAMETER  $tempBarrel
```

```
        VALID WHEN  EXISTS (
                        CREATE OUTPUT SNAPSHOT  temp_of_barrelControl(nodeId ID)
                        WITH DURATION 1h AS
                        LOW:
                                EVERY ONE
                                SELECT ID
                                SAMPLING
                                        EVERY 1h
                                        WHERE temperature > $tempBarrel
                                EXCUTE IF EXISTS (temperature) )
        RELATIVE SENSORS:  temp_of_barrelControl(nodeId ID)
}


CREATE DIMENSION Temp_of_Bottle
SPECIFY In_bottle
{
        WITH PARAMETER  $tempBottle
        VALID WHEN  EXISTS (
                        CREATE OUTPUT SNAPSHOT  temp_of_bottleControl(nodeId ID)
                        WITH DURATION 1h AS
                        LOW:
                                EVERY ONE
                                SELECT ID
                                SAMPLING
                                        EVERY 1h
                                        WHERE temperature >  $tempBottle
                                EXCUTE IF EXISTS (temperature) )
        RELATIVE SENSORS:  temp_of_bottleControl(nodeId ID)
}
```

Creazione di possibili contesti relativi al caso di studio

Esempio 1

Si vuole creare un contesto che rappresenti la situazione in cui un viticoltore che si trova nel vigneto sia interessato alla fase di coltivazione dell'uva.

Egli quindi vorrà monitorare le condizioni ambientali (umidità e temperatura) in cui si trova il vigneto, attraverso l'interrogazione di tutti e solo i sensori presenti al suo interno.

```
CREATE CONTEXT ColtivazioneTime
ACTIVE IF Role = viticoltore AND Situation = In_Vigneto AND Interest_Topic = Coltivazione
ON ENABLE    CREATE OUTPUT STREAM Monitoring (nodeId ID, temperature FLOAT,
             humidity FLOAT, locationX FLOAT, locationY FLOAT) AS
             LOW:
                  EVERY ONE
                  SELECT ID, temperature, humidity, locationX, locationY
                  SAMPLING
                        EVERY 1m
                  EXECUTE IF EXISTS(temperature) OR EXISTS(humidity)
ON DISABLE Drop Monitoring
REFRESH EVERY 1h
```

Esempio 2

In questo secondo esempio si crea un contesto che rappresenti la situazione nella quale un agronomo, oppure un cantiniere, stia controllando le condizioni della cantina (di solito sono queste due categorie di lavoratori che sono addetti a questa mansione).

Non ci si trova però in una situazione "standard": la cantina contiene alcune bottiglie che sono ad una temperatura superiore a quella definita "critica".

Quindi è fondamentale che il lavoratore possa rapidamente rilevare questa situazione, acquisendo oltre alla temperatura anche la collocazione delle "bottiglie a rischio".

Da notare in questo esempio che si usa una dimensione parametrizzata (Temp_of_Bottle = 30°C).

```
CREATE CONTEXT AltaTempInBottle
ACTIVE IF (Role = agronomo OR Role = cantiniere) AND Temp_of_Bottle = 30°C
ON ENABLE CREATE OUTPUT STREAM MonitoringBottle (nodeId ID, temperature FLOAT,
                locationX FLOAT, locationY FLOAT) AS
                  LOW:
                       EVERY ONE
                       SELECT ID, temperature, locationX, locationY
                       SAMPLING
```

```
                    EVERY 1m
ON DISABLE Drop MonitoringBottle
REFRESH EVERY 1h
```

Esempio 3

Questo contesto rappresenta la situazione in cui il vigneto è a rischio gelata durante la fase di coltivazione, che si presume sia nel periodo tra il 21 marzo e il 31 settembre.

Si assume che ci sia rischio gelate ad una temperatura al di sotto dei 5°C e ad un'umidità superiore al 75%.

Sarà utile quindi rilevare temperature, umidità con l'aggiunta della locazione geografica di tutti e soli i sensori che rilevano questa situazione.

Questo contesto è discretizzato da dimensioni appartenenti a tutte e tre le tipologie definite:

- dimensione con nodi-concetto standard (*Role = viticoltore*)
- dimensione con parametro (*CriticalTemp_of_Vigneto = 5°C, Humidity%_of_Vigneto = 75*)
- dimensione con nodo-concetto con parametro (*Season(21/03, 31/9)*)

```
CREATE CONTEXT FrostAlarm
ACTIVE IF Role = viticoltore AND CriticalTemp_of_Vigneto = 5°C AND Humidity%_of_Vigneto
= 75 AND Time = Season(21/03, 31/09)
ON ENABLE   CREATE OUTPUT STREAM MonitoringFrost (nodeId ID, temperature FLOAT,
            humidity FLOAT, locationX FLOAT, locationY FLOAT) AS
            LOW:
                EVERY ONE
                SELECT ID, temperature, humidity, locationX, locationY
                SAMPLING
                        EVERY 1m
ON DISABLE Drop MonitoringFrost
REFRESH EVERY 1h
```

CONSIDERAZIONI FINALI

Una volta creati tutti i contesti possibili per un determinato utilizzatore, occorre realizzare una politica di gestione che sappia risolvere i casi nei quali più di un contesto è attivabile.

Negli esempi presentati, i contesti *ColtivazioneTime* e *FrostAlarm* sono entrambi relativi ad un viticoltore e quindi potenzialmente concorrenti, poiché il primo contesto è quello tipico in cui si trova un viticoltore all'interno del vigneto, mentre il secondo rappresenta un caso particolare, meno frequente, in cui egli si può trovare (vigneto a rischio gelata).

Una possibile soluzione è quella di introdurre, nella sintassi di creazione del contesto, un costrutto dedicato a definirne la priorità di attivazione.

Per esempio se sono tre i contesti possibili per un determinato lavoratore, si potrà effettuare una scelta a priori, stabilendo per ognuno di essi una priorità diversa, ad esempio su una scala da 0 (priorità inferiore) a 2.

Esempio

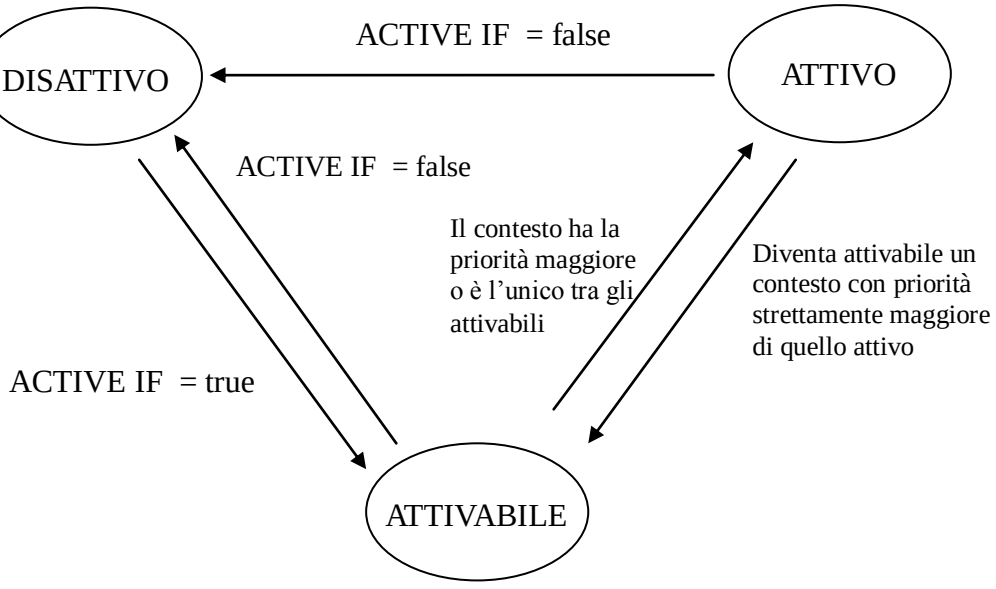
```
CREATE CONTEXT ColtivazioneTime
PRIORITY = 0
ACTIVE IF Role = viticoltore AND Situation = In_Vigneto AND Interest_Topic = Coltivazione
ON ENABLE    CREATE OUTPUT STREAM Monitoring (nodeId ID, temperature FLOAT,
             humidity FLOAT, locationX FLOAT, locationY FLOAT) AS
             LOW:
                  EVERY ONE
                  SELECT ID, temperature, humidity, locationX, locationY
                  SAMPLING
                        EVERY 1m
                  EXECUTE IF EXISTS(temperature) OR EXISTS(humidity)
ON DISABLE Drop Monitoring
REFRESH EVERY 1h

CREATE CONTEXT FrostAlarm
PRIORITY = 1
ACTIVE IF Role = viticoltore AND CriticalTemp_of_Vigneto = 5°C AND Humidity%_of_Vigneto
= 75 AND Time = Season(21/03, 31/09)
ON ENABLE    CREATE OUTPUT STREAM MonitoringFrost (nodeId ID, temperature FLOAT,
             humidity FLOAT, locationX FLOAT, locationY FLOAT) AS
             LOW:
                  EVERY ONE
                  SELECT ID, temperature, humidity, locationX, locationY
                  SAMPLING
                        EVERY 1m
ON DISABLE Drop MonitoringFrost
REFRESH EVERY 1h
```

Una volta che questi due contesti vengono creati, molto probabilmente *FrostAlarm* non sarà attivabile poiché la sua ACTIVE IF risulterà uguale a false, essendo caratterizzato da un evento solitamente raro.
Presumiamo quindi che inizialmente sia attivo *ColtivazioneTime.*
Una volta che ACTIVE IF di FrostAlarm risulta verificata, FrostAlarm sarà attivato a discapito di ColtivazioneTime, poiché la sua priorità risulta strettamente maggiore.

Stati di un contesto

Per le considerazioni precedenti è utile definire le fasi in cui si può trovare un contesto durante il suo ciclo di vita.
Una volta creato, viene valutata la ACTIVE IF, se risulta verificata, il contesto diventa ATTIVABILE, altrimenti viene definito DISATTIVO
Un contesto, se risultasse l'unico o quello con priorità maggiore tra i possibili contesti ATTIVABILI, allora passerebbe da ATTIVABILE ad ATTIVO, attivando così la query presente all'interno della clausola ON ENABLE.
Un contesto ATTIVO può ritornare nello stato ATTIVABILE, se un contesto con priorità maggiore della sua diventa ATTIVABILE.
Un contesto ATTIVO o ATTIVABILE può ritornare allo stato DISATTIVO se la relativa ACTIVE IF perde la sua veridicità.



BIBLIOGRAFIA

[1] A. K. Dey, G. D. Abowd, “Towards a better understanding of context and context-awareness”, Technical Report GITGVU- 99-22, Georgia Institute of Technology, College of Computing, 1999.

[2] Marco Marino, “A proposal for a context-aware extension of PerLa language”

[3] Bolchini C., Quintarelli E., Rossato R.: *Relational data tailoring through viewcomposition*. In: Proc. Intl.Conf.on Conceptual Modeling(ER’2007), Springer LNCS, Vol. 4801, 2007, p. 149-164

[4] C. Bolchini, C.A. Curino, E. Quintarelli, F.A. Schreiber, L. Tanca, “Context information for knowledge reshaping”

[5] Cristiana Bolchini, Giorgio Orsi, Elisa Quintarelli, Fabio A. Schreiber, Letizia Tanca, “Progettazione dei dati con l'utilizzo del contesto”, In: Mondo Digitale n.3-settembre 2008

[6] http://artdeco.elet.polimi.it

[7] Marco Fortunato, Marco Marelli, Tesi di Laurea “Design of a declarative language for pervasive systems”