# Scopes Declarative Language

**Pablo Guerrero, Daniel Jacobi, Ilia Petrov, Alejandro Buchmann**
E-mail: {guerrero,jacobi,petrov,buchmann}@dvs.tu-darmstadt.de

TECHNISCHE
UNIVERSITÄT
DARMSTADT

DVS Databases and Distributed Systems Group

## 1 Introduction

The goal of this document is to explain and exemplify the declarative language used to operate the Scopes framework.

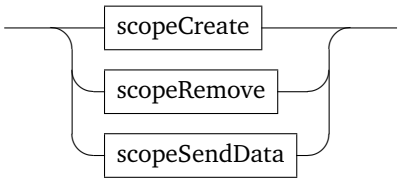
## 2 The Language

We proceed with a top down approach to the Scopes language.

### 2.1 Scope Operations

There are three operations, namely, to create a scope, to remove a scope, and to send data to all of a scope's members.

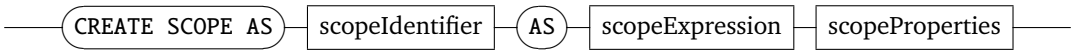
*operation*



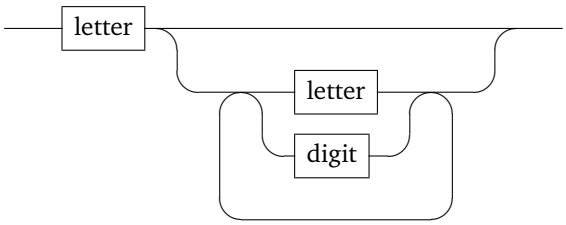

### 2.2 Scope Creation

This is the most complex statement, as it is very expressive and has many keywords. The general structure of the `CREATE` statement is:

*scopeCreate*

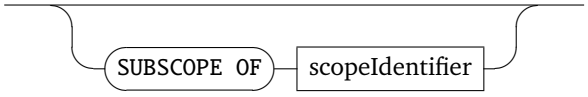



*scopeIdentifier*




A scope can be specified as nested within another, parent scope. Both scope and parent scope are specified by a scope identifier, however, scope identifiers are hashed to 16-bit ids, which are used within the sensor network.

*scopeProperties*

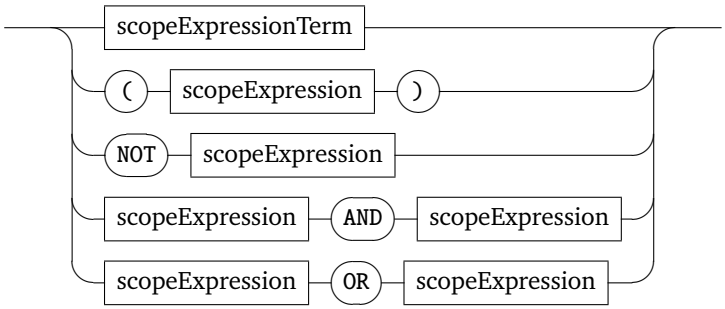



An (incomplete) example can thus be:

```
CREATE SCOPE ChildScope AS (
  ... ) SUBSCOPE OF ParentScope;
```

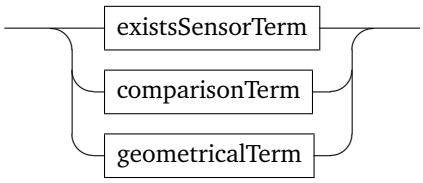
Scopes are defined by means of logical expressions. These can be connected by logical operators `AND`, `OR` and `NOT`, or be recursively defined between parenthesis.

*scopeExpression*



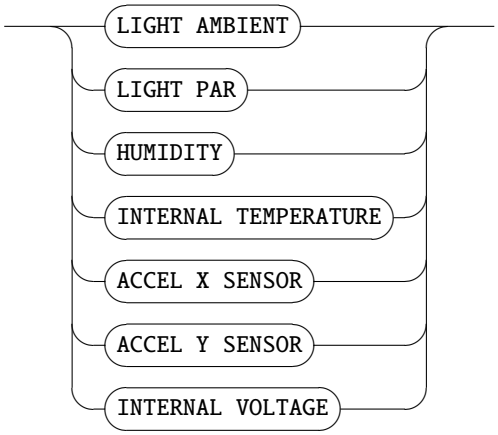

At some point, logical expressions contain terms, e.g., subparts of the expression that evaluate to true or false. In Scopes there are three types of terms. To check whether a node has a populated sensor or not, the `EXISTS SENSOR` term is used (see below the array of predefined sensors). A typical term is that for comparison, which is binary. There are four types of operands that can be used for comparison, Finally, geometrical terms check for a three-dimensional aspect, typically using the position of the node.

*scopeExpressionTerm*

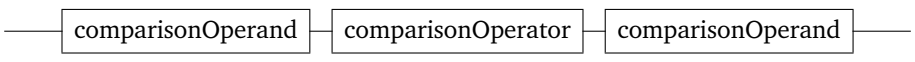



*existsSensorTerm*

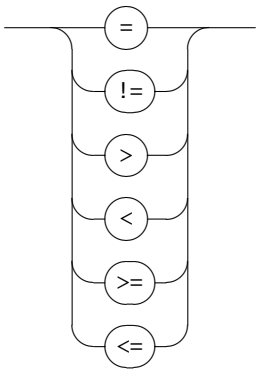

*sensorType*



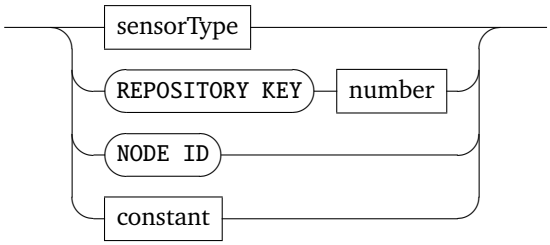

*comparisonTerm*



*comparisonOperator*




*comparisonOperand*




A compact example illustrating the first two clause types is presented next:

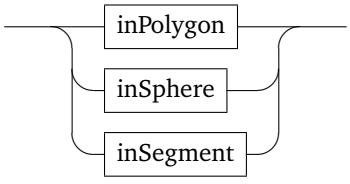
```
CREATE SCOPE ScopeExample AS (
  ( EXISTS SENSOR HUMIDITY AND NOT HUMIDITY <= 70) OR
  NODE ID > 20 OR
  REPOSITORY KEY 7 == 273);
```

This expression is evaluated to true by a node if:

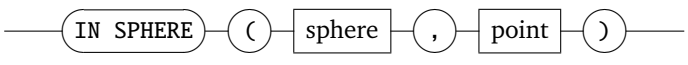
- both the humidity sensor is populated and the humidity is greater than 70, or
- the node's id is greater than 20, or
- the repository entry with key value 7 equals the constant value 273.

There are three geometrical operators supported currently by Scopes. The `IN SPHERE` clause is quite straightforward: it takes a sphere and a point as arguments, and returns true if the point is within or at the borders of the sphere, and false otherwise. Spheres are defined by their center and radius. The `IN SEGMENT` clause takes a segment and a point as argument, and returns true if the point is within the segment, false otherwise. Segments are defined as a sequence of points and a segment width. The `IN POLYGON` clause takes a polygon and a point as an argument, and returns true if the point is inside or at the borders of the polygon. Polygons are defined as a sequence of consecutive points, the last of which is implicitly connected to the first.

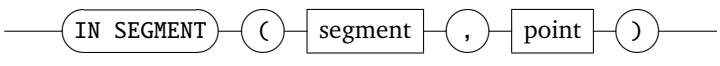
*geometricalTerm*



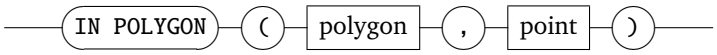

*inSphere*



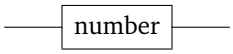

*inSegment*



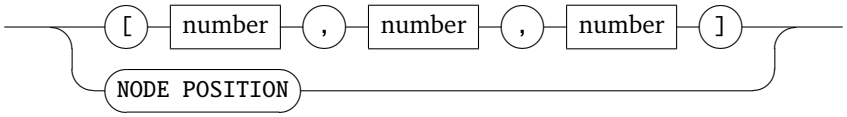
*inPolygon*



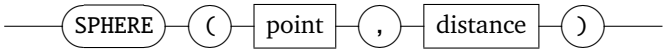

*distance*



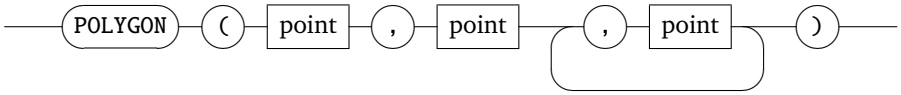

*point*



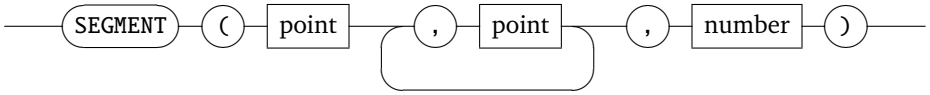

*sphere*




*polygon*




*segment*




As example of these constructs we present the following statement:

```
CREATE SCOPE GeomScopeExample AS (
  IN SPHERE ( SPHERE ( [100,50,50], 30), [101,51,49] ) OR
  IN POLYGON ( POLYGON ([0,0,0], [0,100,0], [50,50,0] ), NODE POSITION) OR
  IN SEGMENT ( SEGMENT ([0,0,0], [50,50,0], [50,100,0], 5)
);
```

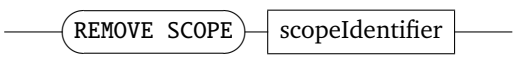
This scope creation evaluates to true if:

- the point [101,51,49] is located within the sphere centered at [100,50,50] and 30 meter radius (`true` for these constants), or
- the node's position is located within the specified triangle (polygon), or
- the node's position is located up to 5 meters from the specified line segments.

## 3 Scope Removal

The removal of a scope is quite straightforward:

*scopeRemove*



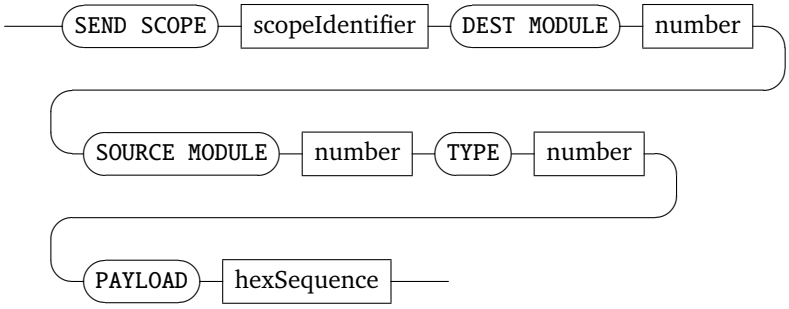

## 4 Sending Data to a Scope

Data can be declaratively sent from a scope's root node to its members. For a connection to an SOS node, the following parameters must be specified:

- `DEST MODULE`: the id of the module that should receive the message

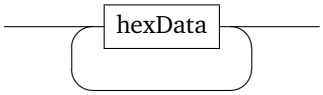
- `SOURCE MODULE`: the module id that should be used as sender module
- `TYPE`: the message type, i.e., the function that should handle the message.

*scopeSendData*




The payload is specified as a sequence of hexadecimal characters with the following format.

*hexSequence*




*hexData*