3.2 An Employee Database

This example illustrates the association between classes, methods, polymorphism and recursion in classes. Continuing with Example 3.1, the company also maintains information about its departments and employees in the database. Some declarations for this database are given below. The Self variable refers to the current object that is being manipulated in a rule:

class employee  
  key: int;  
  name: string;  
  salary: real;  
  make: {part}; /* it relates with Example 3.1, the set 
  * of parts produced by the employee */  
module  
salary_of(Self.salary). /* a class method */  
endclass

class operator isa employee /* inheritance*/  
overtime: int;  
overtime_pay: real;  
module  
salary_of(Self.salary+Self.overtime*Self.overtime_pay).  
endclass

class supervisor isa employee /* inheritance*/  
  bonus: real;  
module  
salary_of(Self.salary+Self.bonus). /* polymorphism*/  
endclass

class department  
  key: int;  
  name: string;  
  chief: supervisor;  
  members: {employee};  
  composed_of: {department}; /* hierarchical structure */  
endclass

Consider the query: "find (the name of) all the employees in a specific department". This can be expressed in SPL as follows:

employees_of(Dept, <E.name>) :-  
  member(E, Dept.members).
The query: "given a part, which employees can manufacture it?" can be formulated in SPL as:

works_on(Part, <E.name>) :-
    employee(E),
    member(Part, E.make).

Note that, because it does not exist a relationship between parts and employees, it is necessary to do a linear search over all employees.

The following rule computes the payroll of one department. The second one is an example of how to invoke methods defined in a class:

department_payroll(Dept, S) :-
    department_pays(Dept, P),
    sum(P, S).

department_pays(D, <X>) :-
    member(E, D.members),
    E.salary_of(X).

4 Query Language

The query language is modular, declarative and provides facilities for recursive rule definition. Rules can be defined using extended Horn clauses. The language also supports lists, tuples, sets and multisets. The invocation of attributes and methods of the objects is direct and is done it using path expressions [14].

4.1 Overview

Rules in SPL are written in the same style as Prolog:

\[ L : L_1, \ldots, L_n \]

where \( L \) (head) is a term and \( L_1, \ldots, L_n \) are (positive or negative) literals, with \( n \geq 0 \); these literals define the body of the rule. This rule can be read in a declarative form as "if \( L_1 \) and \( L_2 \) and \( \ldots \) and \( L_n \) then \( L \)". Each term of the body can include:

- typed simple variables;
- path expressions to make references to attributes;
- class variables to make references to all objects (instances) of a given class;
- tuple constructors.

If \( n = 0 \) in the previous rule then \( L \) is a fact. In SPL variables cannot appear in facts.

The head of the rule can only include: typed simple variables, path expressions to make references to object attributes, sets and multisets. Sets and multisets appear frequently in