Оглавление

Математическая основа d0sl

Теория семантического моделирования (∑-программирования) была впервые предложена тремя знаменитыми советскими математиками в 80-х годах прошлого века (академик Ю.Л. Ершов, академик С.С. Гончаров, д.ф.-м.н. Д.И. Свириденко), и, по версии Американского математического общества, вошла в 100 величайших достижений науки в 20-м веке. Семантические технологии представляют из себя следующий логический шаг в развитии ИТ технологий и позволяют разрешить кризис в развитии технологий искусственного интеллекта (AI – artificial intelligence).

Функциональность d0SL

Платформа d0sl позволяет управлять логикой поведения сложных систем, используя язык d0sl, понятный специалисту в предметной области. Платформа имеет широкое поле применения — от бизнес-процессов предприятия, до управления проектами или поведением автономных систем, включая системы Искусственного Интеллекта и Интернета Вещей.

1. Технология d0SL позволяет создавать спецификации задач на формальном логическом языке, которые преобразуются в исполнимый код без участия человека (используется технология кодогенерации).
2. Язык d0SL — язык логических правил (т.н. предикатов). Язык d0SL спроектирован так, чтоб быть максимально простым, интуитивно понятным и, в то же время, максимально выразительным для создания спецификаций.
3. Язык и технология d0SL основаны на многолетних исследованиях в области математической логики и computer science.
4. Язык d0SL можно легко расширять за счет добавления функций, предикатов и классов объектов доменных моделей. Таким образом d0SL сам по себе является

метаязыком для создания предметно-ориентированных языков (DSL — domain specific languages)

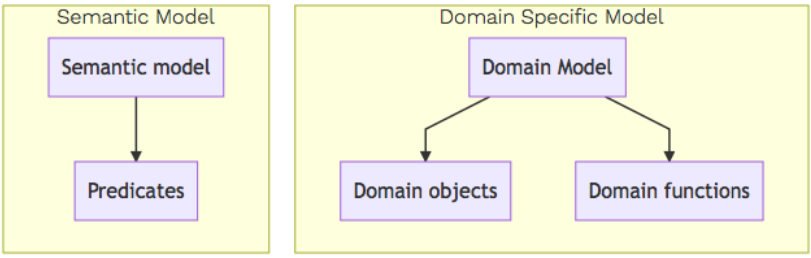
5. Понятия семантических моделей языка d0SL и возможность задания новых семантических предикатов/правил на основе других позволяет сколь угодно много повышать уровни абстракции и таким образом создавать предметно-ориентированные локальные языки для упрощения работы специалистов в данных предметных областях.

Типы документов моделей в d0SL

В d0SL есть два типа документов моделей — Семантические Модели (Semantic Model) и Доменные Модели (Domain Model — аналог сигнатуры в теории моделей)
В семантической модели пользователь задает определения семантических (т.е. определяемых) предикатов через логические конструкции (т.е. формулы / правила)

В доменной модели определяются типы объектов и объявляются доменные функции/предикаты. Эти типы и доменные функции затем могут быть использованы при задании правил в семантических моделях. Реализация доменных функций лежит уже в зоне ответственности профессиональных разработчиков. Доменные модели по сути являются драйверами для интеграции семантических моделей с окружающим миром, например такими как внешними СУБД, системами CRM, нейро-сетями, аппаратными датчиками и т.п. С технической точки зрения

доменная модель является аналогом интерфейса (interface)



языка Java или header файла в C/C++.

Чтоб создать новую семантическую модель или доменную модель, нажмите правой кнопкой мыши на разделе sandbox и выберите new/SemanticLanguage/DomainSpecificModel или new/SemanticLanguage/SemanticModel

Пример семантической модели

```
model ChessAII def
  use ChessDSL from org.d0sl.examples.chess.ChessDSL
  def start() means
    check all
      ChessDSL.start()
    end
  end def
  def check board(board : ChessBoard) means
    check all
      var queens = ChessDSL.get queens(board)
      for all q1, q2 in queens
```

```
        not ChessDSL.on one line(q1, q2) and not
ChessDSL.on one diagonal(q1, q2)
    end
  end def
end ChessAII
```

Пример доменной модели

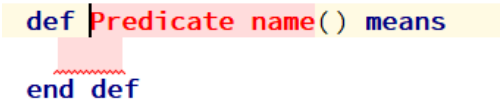
```
domain specific model Math def
    # Math library
    # Square root
    fun sqrt(value : numeric) returns numeric
    # Sinus and Cosinus
    fun sin(value : numeric) returns numeric
    fun cos(value : numeric) returns numeric
    # Power function
    fun pow(value : numeric, power : numeric) returns
numeric
end Math
```

Предикаты

В d0SL используется трехзначная логика: истина, ложь и неизвестно — true, false, none (undefined).

Если формула возвращает none, то вы это должны понимать как исключение, например, что произошла ошибка или таймаут при вычислении этого правила

Для задания нового предиката в семантической модели напишите ключевое слово def, после чего автоматически



появится шаблон определения предиката:
Имена предикатов могут заданы как латинскими так и русскими буквами, содержать пробелы.

Вначале шаблон предиката появляется без аргументов, которые вы можете задать по вашему усмотрению. Для этого поместите курсор между скобками в заголовке предиката и нажмите клавишу Ввод.

В теле (правиле) предиката вы можете задать любую логическую формулу языка d0sl с использованием логических операций: `and`, `or`, `not`, `check all`, `if then`, `for all`.

Однако для простоты мы рекомендуем вам начать задавать правило с оператора `check all`. Оператор `check all` эквивалентен оператору `and` с множеством аргументов и введен для упрощения синтаксиса, чтоб не надо было много раз повторять конструкцию `and`.

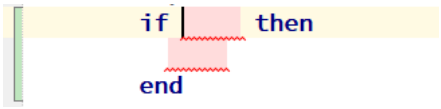
Пример предиката

```
def check board(board : ChessBoard) means
  check all
    var queens = ChessDSL.get queens(board)
    for all q1, q2 in queens
      not ChessDSL.on one line(q1, q2) and not
ChessDSL.on one diagonal(q1, q2)
```

```
  end
end def
```

Импликация (конструкция if then)

Оператор `if` в результате дает `false` только в случае, когда условие равно `true` и при этом следствие (часть после `then`) равно `false`. Введите оператор `if` , а редактор сам вставит



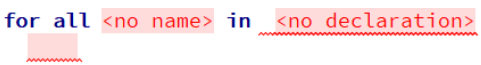
шаблон оператора `if then:`

Пример if

```
if AutodromeDSL.road sign(car) then
  AutodromeDSL.road sign allows move(car)
end
```

Оператор for all

Чтоб задать оператор `for all` напечатайте `for`. После чего появится



шаблон оператора `for all:`
Общая форма этого оператора
`for all` x_1, x_2... x_n `in list`$_X$ `Expression(`x_i`)`

Это означает, что будет проверяться, что для всех $x_1, x_2 ... x_n$ из списка $list_X$ выражение $Expression(x_1)$ and … and

```
var queens = ChessDSL.get queens(board)
for all q in queens
  not for all qq in queens
    not ChessDSL.on near line(q, qq)
for all q1, q2 in queens
  not ChessDSL.on one line(q1, q2) and not ChessDSL.on one diagonal(q1, q2)
```

$Expression(x_n)$ должно быть истинно (true).

Переменные

Вы можете задавать переменные введя оператор `var` внутри блока `check all`

Важное ограничение. В отличие от обычных языков программирования:

1. Вы не можете изменить (переприсвоить) значение переменной после того, как первый раз задали
2. Каждый раз, когда вычисляется правило (тело предиката), значение переменной вычисляется только один раз.

Пример

```
# Testing for sin & cos
def test() means
  check all
    var angle = 35
    var cosinus = Math.cos(angle)
    var sinus = Math.sin(angle)
    var sum of squares = Math.pow(cosinus, 2) +
Math.pow(sinus, 2)

    # considering the features of the library
java.lang.Math
    # and inaccurate calculations when converting
degrees to radians
    sum of squares <= 1
    sum of squares >= 0.999999
  end
end def
```

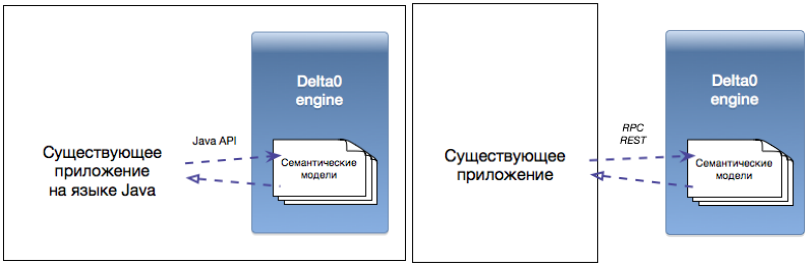
Подсказки в редакторе

Комбинации	Действие
Ctrl+space	Вызов подсказки для редактирования
Alt+enter	Меню
Ctrl+w	Выбор элемента
def	Задать предикат
use	Задать оператор `use`
check	Задать оператор Check all
and, or, not	Логические операции
“	””, задать строку
for	`For all`
if	if then
Insert, enter	Для вставки нового аргумента функции или предиката
var	Задание локальных переменных внутри блока check all
fun	Задать функцию в Domain Model

typedef	Задать новый тип в Domain Model

Место семантического моделирования в архитектуре ПО

Платформа d0sl позволяет реализовать часть или всю бизнес логику в виде семантических моделей на языке Delta0 (d0sl). Архитектура семантического движка реализована с учетом того, чтоб его можно было встраивать внутрь системы, или же использовать как внешний сервер «логических приложений» (то



есть семантических моделей).

Жизненный цикл

Если упростить, то в классическом подходе аналитик создаёт ТЗ/спецификации, на основе которых уже разработчики делают архитектуру и код.

Опять же упрощенно при использовании семантического моделирования аналитик создает исполнимую спецификацию на языке Delta0. В языке Delta0 есть исключительно логические конструкции (логические операции И, ИЛИ, НЕ, ЕСЛИ ТО и перебор по конечным спискам), в тоже время в нем отсутствуют какие-бы то ни было процедурные конструкции и он прост для освоения (требуется около одного дня для освоения).

Это позволяет снизить затраты на разработку логики приложения за счет того, что модель, созданная аналитиком, может быть исполнена сразу. Кроме того, модель можно привязать не сразу к рабочему окружению, а вначале к тестовому (конструкция USE в языке d0sl). В частности, это позволяет делать имитационное семантическое моделирование для проверки/тестирования моделей.

Семантическое моделирование позволяет вносить изменения в бизнес логику уже на этапе эксплуатации без привлечения разработчиков.

Схема работы с d0sl SDK

Аналитик работает в среде JetBrains MPS с загруженным плагином d0sl.

Он в этой среде может создавать и тестировать свои семантические модели на языке Delta0. Эта среда позволяет:

1. использовать системы контроля версий;
2. исполнять модели в настроенном тестовом окружении;

3. развертывать модели в рабочем окружении через используемую в компании систему continuous integration.

Основные принципы создания семантических моделей

1. Семантическая модель состоит из набора определений предикатов.
2. Предикаты могут вызываться из внешних приложений (в некотором смысле событийная модель).
3. В d0sl нет никаких ленивых вычислений, предполагаются только явные вызовы предикатов.
4. Делайте предикаты как можно более понятными для других аналитиков/специалистов предметной области:
    a. Именуйте предикаты разумно и понятно (см п. 6 ниже);
    b. Делайте определения предикатов понятными;
    c. Используйте для этого в своем процессе разработки peer-review;
    d. Старайтесь делать короткие определения предикатов, лучше декомпозируйте (см ниже);
    e. Старайтесь не делать более одного уровня вложенности конструкций IF .. THEN. Если у вас два или более уровня, попробуйте понять то что

вы написали сами через неделю и проверьте поймет ли вашу спецификацию ваш коллега без ваших объяснений.

5. Помните, что в d0sl нет состояний, сам семантический движок stateless

    a. Если вам нужно работать с состояниями, используйте свои доменные модели для этого;

    b. Данные в модель могут попасть только из двух источников:

        i. Параметры вызова предиката;

        ii. или то, что вы можете получить из доменных функций;

    c. Следствие такого подхода: для вычисления предиката ему совсем не обязательно иметь все данные, ему достаточно того, что возвращают доменные функции и того, что ему передали в качестве параметров;

    d. Это позволяет защищать данные, оставляя на месте те из них, которые особо ценны.

6. Декомпозиция сверху вниз, от юзкейсов и их критериев, с учетом пред- и постусловий:

    a. При декомпозиции не заводите искусственные сущности, руководствуйтесь тем, как вы это описываете на своем родном языке;

    b. При декомпозиции задавайте себе вопросы «что это значит?» и «какие критерии для достижения такого результата?».

7. В d0sl нет рекурсий. Если уж вам кажется что они вам нужны, лучше используйте кванторы по конечным спискам (перебор).

8. Мы пока решили обойтись без раздела ИНАЧЕ (ELSE) в конструкции IF .. THEN , поскольку в речи мы практически не пользуемся такой конструкцией.

Основные принципы диагностики проблем и отладки семантических моделей

Поскольку семантическое моделирование основано на идее исполнимых декларативных логических спецификаций, то и для диагностики проблем должна использоваться логика. Как можно тестировать свою семантическую модель:

1. Для своей семантической модели вы можете сделать тестовую семантическую модель.

2. Или тестовые предикаты в самой своей модели.

3. Кроме того, помните про предусловия и постусловия, которые следует явно специфицировать в своей модели.

4. Используйте диагностический вывод на экран.

5. Старайтесь все специфицировать как можно более явно (explicit). В самом движке d0sl мы, как нам кажется, закрыли возможности для неявного поведения.

Ссылки

1. Гончаров С.С., Ершов Ю.Л., Свириденко Д.И. Методологические аспекты семантического программирования // Научное знание: логика, понятия, структура. - Новосибирск, Наука 1987. - с. 154-184.
2. В.Ш. Гумиров, "Объектно-ориентированный вариант языка Σ-спецификаций," [Online]. Available: https://goo.gl/UjvUrp
3. V.Gumirov, P.Matyukov, D.Palchunov, Semantic Domain Specific Languages, IEEE, 2018 (перевод препринта на русский язык доступен http://bit.ly/sDSL-RU)
4. Пальчунов Д.Е. Решение проблемы извлечения информации на основе онтологий // Бизнес-информатика, №1, 2008, — стр. 3–13
5. S.S. Goncharov, D.I. Sviridenko Σ-Programming Amer. Math. Soc. Transl. (2) Vol. 142, 1989. Available: https://goo.gl/QcocUc
6. Сайт проекта https://d0sl.eyeline.ru/