АЛГЕБРО-ЛОГИЧЕСКИЕ МЕТОДЫ В ИНФОРМАТИКЕ И ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

ALGEBRAIC AND LOGICAL METHODS IN COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE

Серия «Математика»
2022. Т. 42. С. 121—137

Онлайн-доступ к журналу:
http://mathizv.isu.ru

ИЗВЕСТИЯ
Иркутского
государственного
университета

Научная статья

УДК 004.5
MSC 68U35, 68N15
DOI https://doi.org/10.26516/1997-7670.2022.42.121

Объектные модели как микросервисы: язык запросов

Д. Н. Гаврилин[1], **И. А. Кустова**[2], **А. В. Манцивода**[1]✉

[1] Иркутский государственный университет, Иркутск, Российская Федерация
[2] ООО «Логус», Иркутск, Российская Федерация
✉ ontobox@ontobox.ru

Аннотация. Нами разрабатывается технология создания приложений на основе расширенного объектно ориентированного подхода, в котором объектные модели обогащены до функционала микросервисов. Такая вариация называется микросервисным объектно ориентированным программированием. Объектная модель как микросервис представляет собой автономную информационную систему, имеющую долговременную память и взаимодействующую с другими моделями и внешними акторами через API. Кроме того, реализация объектных моделей как микросервисов обеспечивает создание принципиально новой low-code-технологии, основанной на визуализации работы с объектными моделями.

В данной работе представлен язык запросов к объектным моделям, эффективно работающий с долговременными моделями. Он является подмножеством языка Libretto. Описывается логическая семантика языка, а также метод его компиляции в SQL, основанный на использовании промежуточного реляционного языка. Компилятор языка запросов в SQL в настоящее время реализован в рамках платформы Ontobox. Он показал высокую эффективность и сегодня активно используется при решении прикладных задач.

Ключевые слова: low-code, микросервисное объектно-ориентированное программирование, язык запросов, Ontobox

Ссылка для цитирования: Гаврилин Д. Н., Кустова И. А, Манцивода А. В. Объектные модели как микросервисы: язык запросов // Известия Иркутского государственного университета. Серия Математика. 2022. Т. 42. С. 121–137.
https://doi.org/10.26516/1997-7670.2022.42.121

Research article

Object Models as Microservices: a Query Language

Denis N. Gavrilin[1], Irina A. Kustova, Andrei V. Mantsivoda[1]✉

[1] Irkutsk State University, Irkutsk, Russian Federation
[2] Logus Ltd., Irkutsk, Russian Federation
✉ ontobox@ontobox.ru

Abstract. We are designing an application development technology based on an extended object-oriented approach, in which object models are enriched to the functionality of microservices. This variation is called microservice object-oriented programming. An object model as a microservice is an autonomous information system that behaves as a persistent storage and interacts with other models and external actors through the API. Moreover, understanding object models as microservices provides a fundamentally new low-code technology based on model visualization.
This paper presents a query language for object models that works efficiently with persistent models. It is a subset of the Libretto language. The logical semantics of the language is described, as well as the method of compiling it into SQL, based on the use of an intermediate relational language. A SQL query language compiler has been implemented within the Ontobox platform. It shows high efficiency and is now actively used in real-life application development tasks solving.

Keywords: low-code, microservice object-oriented programming, query language, Ontobox

For citation: Gavrilin D. N., Kustova I. A., Mantsivoda A. V. Object Models as Microservices: a Query Language. *The Bulletin of Irkutsk State University. Series Mathematics*, 2022, vol. 42, pp. 121–137. (in Russian)
https://doi.org/10.26516/1997-7670.2022.42.121

1. Введение

В последние годы качественно изменился механизм проникновения IT-технологий в управленческую и коммерческую деятельность – вместо IT-поддержки отдельных компонент идет переход к тотальной цифровизации, которая погружает в цифровое пространство целые компании. Это ужесточает требования к приложениям, реализующим цифровизацию. Сегодня они должны уметь сочетать операционное управление с бизнес-аналитикой и искусственным интеллектом, быть гибкими и адаптивными к изменениям в волатильной внешней среде.

Кроме того, происходит резкое увеличение объемов работы, что приводит к серьезному дефициту кадров в среде IT-разработчиков. В результате стали активно развиваться и поддерживаться технологии, способные качественно (на порядок) повысить производительность труда при разработке приложений и расширяющие круг разработчиков. Одним из основных трендов здесь является технология low-code [5]. Развивается большое количество проектов, ряд компаний преодолели планку единорогов. С другой стороны, low-code, несмотря на очевидную пользу [8], имеет серьезные трудности в развитии [3; 10]. С нашей точки зрения, эти трудности обусловлены тем, что в современных low-code-платформах используется технология, основанная на диаграммах описания бизнес-процессов (см. спецификацию BPMN, [4]), у которой отсутствует как декларативная, так и внятная процедурная семантика. Кроме того, эта технология строится изолированно от традиционного программирования (high-code), что не позволяет опереться на мощные решения, наработанные в программировании за многие годы развития. Это доставляет большие проблемы профессиональным разработчикам, часто сталкивающимся с необходимостью сочетания low-code с high-code традиционного программирования.

Поэтому наша группа реализует low-code-технологию, основанную на принципиально иных подходах. Идея заключается в том, чтобы решать имеющиеся проблемы, не выходя за рамки традиционного программирования с целью не только обеспечить преимущества low-code, но и избавиться от его текущих недостатков. Наш low-code строится не в стороне от традиционного программирования, а как визуальная надстройка над ним.

Кроме повышения на порядок производительности труда и расширения круга разработчиков, нашими целями являются:

– бесшовное сочетание low-code- и high-code-технологий,

– обеспечение интеграции операционной работы, бизнес-аналитики и искусственного интеллекта,

– открытость приложений к модернизации и развитию.

Интеграция low-code и high-code дает синергию преимуществ обоих уровней — визуальной разработки и традиционного текстового кодирования. При этом визуальные инструменты low-code не изолированы, а являются надстройкой, интерфейсом к high-code [1].

В качестве high-code в нашем проекте используется объектно ориентированный подход (ООП). В процессе работы оказалось, что для успешной реализации поставленных задач необходимо переосмыслить инструментарий ООП. Это касается как языка программирования, так и центрального понятия ООП — объектной модели. Поэтому нами существенно развита концепция объектных моделей [2], превращающая

объектную модель в микросервис [9]. В этом подходе объектная модель – автономная информационная система, имеющая долговременную память и взаимодействующая с другими моделями и внешними акторами через API. Объектные модели также обогащены инструментами управления жизненным циклом объектов и бизнес-процессов [7]. Для поддержки этой версии объектных моделей и построения low-code на основе ООП нашей группой разработан собственный язык программирования Libretto. Сам подход мы назвали микросервисным объектно ориентированным программированием. Он реализован в рамках платформы Ontobox (ранее bSystem).

Ключевой особенностью объектной модели как микросервиса является её функционирование как автономного долговременного хранилища данных. Эта публикация посвящена реализации объектных моделей как долговременных хранилищ. Как известно, эффективная работа с данными в БД основана на использовании языка запросов, например, SQL в реляционных СУБД. Выразительный и эффективный язык запросов существенно повышает и качество процесса разработки, и качество самого приложения. Поскольку в отличие от реляционных БД объектные модели имеют графовую архитектуру, то и язык запросов к ним должен иметь навигационную природу, позволяющую строить пути к искомым данным в этом графе. Семантика языка подобного рода рассматривалась в работе [6].

В данной работе представлен язык запросов к объектным моделям, эффективно работающий с долговременными моделями. Этот язык является подмножеством языка Libretto. Описывается логическая семантика языка, а также метод компиляции языка в SQL, основанный на использовании промежуточного реляционного языка. Компилятор языка запросов в SQL в настоящее время реализован в рамках платформы Ontobox. Он показал высокую эффективность и сегодня активно используется в реализации прикладных задач.

2. Определения

В работе используется упрощенный вариант формализации моделей из [7]. Пусть $\mathcal{B} = \langle\, B_1, \ldots, B_l; \Omega \,\rangle$ – многоосновная алгебраическая система, интерпретируемая как система базовых типов данных с основными множествами B_i. Последовательностью элементов назовем выражения вида $(e_1, \ldots, e_m)$ со следующими свойствами:

$$(e) = e, \qquad (\ldots, (e_1, \ldots, e_m), \ldots) = (\ldots, e_1, \ldots, e_m, \ldots).$$

Первое равенство позволяет не отличать одноэлементную последовательность и сам элемент. Второе равенство говорит о том, что после-

довательности являются плоскими (отсутствует вложимость). Допускаются пустые последовательности ().

Введем понятие документной модели $\mathcal{M}$. Документная модель — упрощенная версия объектной модели без иерархии классов (наследования). $\mathcal{M}$ определяется как

$$\mathcal{M} = \langle\, B_1, \ldots, B_l, O, S; \Omega, C, F \,\rangle. \tag{2.1}$$

Здесь $O = \{o_1, \ldots, o_m\}$ — конечное множество элементов, которые будем называть объектами модели $\mathcal{M}$, S — множество произвольных последовательностей элементов объединенного множества $B_1 \cup \ldots \cup B_l \cup O$, C – набор одноместных предикатных символов, называемых именами классов, а F – набор двухместных предикатных символов, называемых именами полей. Все элементы множеств O и B_i являются в модели $\mathcal{M}$ выделенными. Множество объектов O разбито на непересекающиеся подмножества $\mathbf{c}_1, \ldots, \mathbf{c}_k$, которые называем классами. $c_i \in C$ интерпретируется как одноместный предикат, выделяющий элементы одноименного класса. Имена полей $f \in F$ интерпретируются как двухместные предикаты $f(o, s)$, первый аргумент которых всегда объект, а второй – непустая последовательность, причем для каждого предиката f:

1) существует класс c, такой, что $\mathcal{M} \models \forall o.\, (\exists s. f(o, s) \supset c(o))$;

2) существует класс c' (или базовое множество B_i), такой, что для любых $o \in O$ и $s \in S$, если $\mathcal{M} \models f(o, s)$, то все элементы s принадлежат классу c' (множеству B_i).

В этом случае будем говорить, что f – это поле класса c, а класс c' (соответственно, множество B_i) определяет тип его значений.

3. Язык запросов

Введем основное понятие данной работы — «язык запросов к модели $\mathcal{M}$». Для сокращения технических деталей дадим упрощенное определение, содержащее основные конструкты. Начнем с определения понятий выборки и условия.

Определение 1 (Выборка)**.** Первичной выборкой *называется*

- *терм* $c()$*, где* c *– имя класса модели* $\mathcal{M}$*,*
- *последовательность объектов* $(o_1, \ldots, o_k), k \geq 0$*, причем существует* c*, такое, что для каждого* o_i $\mathcal{M} \models c(o_i)$*.*

Выборка *определяется следующим образом:*

1) *Первичная выборка является выборкой.*

2) *Константа контекста* \$ *является выборкой.*

3) *Если* s *— выборка и* f *– имя поля, а* u *— условие, то* $(s.f)$ *и* $(s.where(u))$ *— выборки.*

Точка является левоассоциативной операцией: $((f_1.f_2).f_3) = f_1.f_2.f_3$. Выборка $f_1.\ \ldots\ .f_n$, $n \geq 1$, называется *простой*, если ни один из f_i не имеет вида $where(u)$.

Определение 2 (Условие)**.** *Атомарным условием называются выражения вида* $s_1 \star s_2$, *где* s_1, s_2 *– простые выборки, а* $\star \in \{=, \neq, >, <, \leq, \geq, \in\}$. *Атомарное условие является условием. Если* A, B *– условия, то* $\neg A, A \wedge B, A \vee B$ *– условия.*

Определение 3 (Запрос)**.** *Выражение*

$$f_1 \,.\, f_2.\ \ldots\ .f_n$$

называется запросом, *если* $f_1 \,.\, f_2.\ \ldots\ .f_n$, $n \geq 1$, *– выборка, а* f_1 *– первичная выборка.*

Определенный выше язык запросов является подмножеством языка Libretto и соответствует его семантике. Пример запроса: найти место работы всех людей, у которых есть внуки младше 10 лет:

```
Person(). where(child. child. age < 10). affiliation   (3.1)
```

Логическая семантика данного запроса может быть выражена как множество:

$$\{a \mid \exists\, p\, c_1\, c_2.\ Person(p) \wedge child(p, c_1) \wedge child(c_1, c_2) \wedge age(c_2) < 10 \wedge affiliation(p, a)\}$$

Опишем семантику запросов на $\mathcal{M}$ с помощью оператора $\texttt{Eval}(ctx, s)$, возвращающего значение пути s в контексте ctx. Введем дополнительное значение $\perp$, будем называть его неопределенным значением. В общем случае значениями запросов s на модели $\mathcal{M}$ являются множества (выборки) элементов $\mathcal{M}$ (как базовых, так и объектов).

1) $\texttt{Eval}(ctx, \perp) = \perp$ для любого ctx.

2) $\texttt{Eval}(ctx, c()) = (e_1, \ldots, e_k)$, где $\{e_1, \ldots, e_k\}$ – все объекты класса c (т.е. $\mathcal{M} \models c(e)$ тогда и только тогда, когда $e \in \{e_1, \ldots, e_k\}$).

3) $\texttt{Eval}(ctx, (e_1, \ldots, e_k)) = (e_1, \ldots, e_k)$ для любой последовательности элементов модели $\mathcal{M}$, $k \geq 0$, и любого ctx.

4) $\texttt{Eval}(e, f) = \perp$, если контекст e – либо значение базового множества B_i, либо объект класса, для которого поле f не определено.

5) $\texttt{Eval}(o,f)=(e_1,\ldots,e_k), k\geq 0$, если o — объект и $\mathcal{M}\models f(o,(e_1,\ldots,e_k))$. Если для любой последовательности $(e_1,\ldots,e_k))$ не выполняется $\mathcal{M}\models f(o,(e_1,\ldots,e_k))$, то $k=0$, то есть $\texttt{Eval}(o,f)=()$.

6) $\texttt{Eval}(ctx,\$)=ctx$ для константы контекста.

7) $\texttt{Eval}(ctx,e.f)=\texttt{Eval}(\texttt{Eval}(ctx,e),f)$

8) Для последовательности элементов $(e_1,\ldots,e_k)$:

$$\texttt{Eval}((e_1,\ldots,e_k),f)=(\texttt{Eval}(e_1,f),\ldots,\texttt{Eval}(e_k,f))$$
$$\texttt{Eval}((e_1,\ldots,e_k),where(u))=$$
$$=(\texttt{Eval}(e_1,where(u)),\ldots,\texttt{Eval}(e_k,where(u)))$$

В частности, $\texttt{Eval}((),s)=()$ для любого пути s.

9) Для элемента e и условия u выполняется

- $\texttt{Eval}(e,where(u))=e$, если u истинно в контексте e.
- $\texttt{Eval}(e,where(u))=()$, если u ложно.
- $\texttt{Eval}(e,where(u))=\bot$, если $u=\bot$.

Понятие истинности условия u в контексте элемента e определяется по индукции.

1) Если u — атомарное условие вида $s_1\star s_2$, где $\star\in\{=,\neq,>,<,\leq,\geq,\in\}$, и $\texttt{Eval}(e,s_1)=e_1$, $\texttt{Eval}(e,s_2)=e_2$, то атомарное условие истинно, если аргументы e_1,e_2 корректны для условия и предикат $e_1\ \star\ e_2$ истинен. Атомарное условие ложно, если аргументы корректны, а предикат $e_1\ \star\ e_2$ ложен. Если хотя бы один аргумент некорректен, то $u=\bot$.

2) Истинность условий $\neg A, A\wedge B, A\vee B$ определяется традиционно с учетом того, что если хотя бы одно из A и B равно $\bot$, то и все условие равно $\bot$.

Условие корректности аргументов e_1,e_2 для $\star\in\{=,\neq,>,<,\leq,\geq\}$: e_1 и e_2 должны содержать ровно одно значение. Условие корректности для $\in$: e_1 должен содержать ровно одно значение, e_2 может быть произвольной последовательностью.

Значение запроса s на модели $\mathcal{M}$ определяется как значение

$$\texttt{Eval}(\bot,s).$$

4. Реляционное представление документных моделей

В рамках развития концепции документных, и более обще, объектных моделей нами были сделаны их несколько реализаций. Долгое время использовалась наша собственная система хранения типа "ключ-значение". Однако в дальнейшем был осуществлен переход к реализации, использующей отображение графовых структур объектных моделей в реляционные базы данных, поскольку, во-первых, в реляционных БД за десятилетия развития накоплен огромный объем наработок, связанный с безопасностью, масштабированием и другими технологическими достижениями и, во-вторых, реляционные базы намного более привычны и приемлемы для пользователей, чем самодельные хранилища. Сегодня основная реализация, используемая в индустриальной версии платформы Ontobox, базируется на СУБД PostgreSQL с возможностью перехода на другие СУБД (Oracle, Derby и др.).

Поскольку графовая архитектура моделей и реляционные данные существенно отличаются, задача погружения моделей в реляционную БД и эффективная реализация языка запросов превращается в нетривиальную задачу. Реальная реализация довольно сложна, насыщена техническими деталями и оптимизациями. Поэтому в данной работе представлена существенно упрощенная её версия, содержащая концептуальные детали.

Реляционной таблицей T размерности $n \times m$ над моделью $\mathcal{M}$ назовем множество кортежей длины n

$$T = \{\langle\, e_1^1, \dots e_n^1 \,\rangle, \dots, \langle\, e_1^m, \dots e_n^m \,\rangle\},$$

где e_i^j — либо пустые последовательности () (аналог NULL), либо элементы базовых типов модели, либо уникальные идентификаторы $\nu(o)$ объектов модели, определенные ниже. Кортежи $r \in T$ называются строками таблицы. i-ю координату кортежей $r \in T$ назовем i-ым столбцом таблицы. Значение i-го столбца в j-й строке T обозначается e_i^j, а значение всего i-го столбца $T^i = \{e_i^1, \dots, e_i^m\}$, $1 \le i \le n$.

Для отображения документной модели в реляционную нам потребуются следующие функции, действующие на документной модели:

- $\nu(o)$ – уникальный идентификатор объекта o модели. Каждому объекту o модели присваивается уникальный $id = \nu(o)$. Для ν определена обратная функция, возвращающая объект по его уникальному идентификатору: $\nu^{-1}(id) = o$;
- $objects(c) = \{\nu(o) \mid \mathcal{M} \models c(o)\}$ – множество идентификаторов всех объектов класса $c \in C$.

Отображаем документную модель в систему реляционных таблиц. В ней объекты классов представляются в виде таблиц с одним столбцом,

содержащим уникальные id объектов класса $c \in C$. Таким образом, объекты класса c модели представлены с помощью таблицы вида

$$T_c = \{\langle\, id_1 \,\rangle, \ldots, \langle\, id_k \,\rangle\}$$

где $objects(c) = \{id_1, \ldots, id_k\}$. Таблица, описывающая значения поля f для всех объектов класса c, содержит два столбца – id объекта и одно из значений поля объекта (которых, по определению полей, может быть несколько). Если поле f объекта id содержит последовательность значений $(e_1, \ldots, e_k)$, то таблица будет содержать k строк $\langle\, id, e_1 \,\rangle, \ldots, \langle\, id, e_k \,\rangle$, $k \geq 0$. Таким образом,

$$T_f = \{\langle\, id, e \,\rangle | id \in objects(c), \mathcal{M} \models f(\nu^{-1}(id), (e_1, \ldots, e_k)); e \in (e_1, \ldots, e_k)\},$$

называется таблицей поля f класса c.

Определение 4 (Реляционное представление)**.** *Реляционным представлением $R_{\mathcal{M}}$ модели $\mathcal{M}$ назовем множество реляционных таблиц, содержащее таблицу T_c для каждого класса $c \in C$ и таблицу T_f для каждого поля $f \in F$.*

Считается, что на представлении $R_{\mathcal{M}}$ нам доступны функции $t(c) = T_c$ и $t(f) = T_f$, возвращающие таблицы для класса c и поля f соответственно.

5. Промежуточный язык

Процесс компиляции языка запросов объектных моделей в SQL состоит из двух этапов. На первом запрос транслируется в терм на промежуточном языке, имеющем, в отличие от первичного запроса, реляционную семантику. На втором этапе терм на промежуточном языке переводится в код на SQL, где функции промежуточного языка интерпретируются как параметрические макросы сегментов кода на SQL. В этом разделе мы определим промежуточный язык и его семантику.

Пусть $\mathcal{M}$ — модель вида (2.1). Определим промежуточный язык, который состоит из условий, выборок и запросов.

Определение 5 (Условие)**.** *Пусть x – переменная. Термами условий являются*

- *элементы базовых типов B_i,*
- *уникальные идентификаторы объектов $\nu(o)$,*
- *термы вида $column(x, m)$, где m – целое число.*

Атомарными условиями являются булева константа тождественной истинности true, а также выражения вида $t_1 \star t_2$ *, где* $\star \in \{=, \neq, >, <, \geq, \leq, \in\}$*, а* t_1, t_2 *– термы условий.*

Атомарные условия являются условиями. Если A *и* B *– условия, то* $A \wedge B$*,* $A \vee B$ *и* $\neg A$ *– условия.*

Каждое условие содержит не более одной переменной x. Данная переменная в качестве значений будет принимать строки таблиц. Мы готовы определить понятия выборки и запроса на промежуточном языке.

Определение 6 (Выборка и запрос на промежуточном языке)**.** *Если* c *— имя класса, то* $t(c)$ *— выборка. Константа контекста* \$ *также выборка. Если* Q *— выборка,* n *— целое,* $f \in F$ *— имя поля, а* P *— условие, то* $Join(Q, n, f)$ *и* $Where(Q, P)$ *— также выборки.*

Если Q *— выборка и* m *— целое, то* $Select(Q, m)$ *— запрос.*

В определениях ниже e (возможно, с индексами) принимают значения базовых типов и идентификаторов объектов модели $\mathcal{M}$. С помощью $P(s)$ будем обозначать условие P, наложенное на строку (кортеж) s таблицы, в котором вместо вхождений переменной x подставлен s.

Определим теперь семантику промежуточного языка. Результат запроса на промежуточном языке E на реляционной модели $R_{\mathcal{M}}$ обозначим через

$$\mathtt{Eval}_R(E).$$

Семантика базовых предикатов условий определяется стандартно, $column(s, m)$ возвращает e_i — значение i-й координаты кортежа s. Истинность условия P на кортеже s обозначим $\models P(s)$.

Значением выборок на промежуточном языке будут произвольные таблицы вида $T = \{\langle\, e_1^1, \ldots e_n^1 \,\rangle, \ldots, \langle\, e_1^m, \ldots e_n^m \,\rangle\}$. В качестве результата запроса $Select(Q, m)$ будет возвращать множество уникальных идентификаторов объектов, которые интерпретируются как ответ на запрос. Операции промежуточного языка преобразуют таблицы T следующим образом:

$$Join(T, m, f) = \{\langle e^1 \ldots e^{k+1} \rangle \mid \langle e^1 \ldots e^k \rangle \in T, \langle e^m, e^{k+1} \rangle \in T_f, 1 \leq m \leq k\}$$
$$Where(T, P) = \{\langle e^1 \ldots e^k \rangle \mid \langle e^1 \ldots e^k \rangle \in T, \models P(\langle e^1 \ldots e^k \rangle)\}$$
$$Select(T, m) = \{e^m \mid \langle\, e^1, \ldots, e^k \,\rangle \in T\}, \quad 1 \leq m \leq k$$

Операция $Join$ добавляет в таблицу T столбец со значениями поля f на объектах, идентификаторы которых находятся в столбце с номером m таблицы T. Фильтр $Where$ сохраняет только те строки таблицы, на которых выполняется условие P. Операция $Select$ по заданной таблице T и номеру столбца m возвращает в качестве результата все значения m-го столбца таблицы T.

Определим оператор $\mathbb{T}(Q)$ перевода запроса Q в выражение на промежуточном языке. При переводе учитываем следующую связь между запросом и соответствующим ему термом на промежуточном языке. Каждый шаг перехода по полю в запросе соответствует одному применению операции $Join$, а каждое применение фильтра $where$ соответствует вызову одноименной операции $Where$. По семантике промежуточного языка каждое применение $Join$ ведет к добавлению $(k+1)$-го столбца в результирующей таблице. Фильтр $Where$ не изменяет число столбцов, но сокращает в общем случае число строк (кортежей) в результирующей таблице в соответствии с условием P. Функция $Select$ задает номер m столбца, в котором содержится результат выполнения запроса.

Таким образом, вычисление запроса на промежуточном языке состоит в монотонном добавлении столбцов и вычищении строк, не соответствующих условиям. Операция $Select$ и термы условий $column(s, n)$ из $Where$ зависят от номеров столбцов, поэтому в процессе определения $\mathbb{T}(Q)$ нам нужно отслеживать номера столбцов, соответствующих нужным нам значениям. Будем отслеживать n – номер крайне правого столбца таблицы (т. е. текущее число столбцов) и n_{cur} — номер столбца, содержащего значения последнего обработанного шага текущего пути.

Заметим, что в зависимости от первичной выборки запрос имеет либо вид $c().p$, либо $(o_1, \ldots, o_k).p$, где p — путь. Поскольку, по определению первичной выборки, $o_1, \ldots, o_k$ должны принадлежать одному классу c, то второй вариант сводится к первому с помощью следующего преобразования:

$$(o_1, \ldots, o_k).p = c().where(\$ \in (o_1, \ldots, o_k)).p$$

Поэтому нам достаточно рассмотреть первый случай.

Для запроса $c().p$ определим

$$\mathbb{T}(c().p) = Select(Where(\mathbb{TP}(p, t(c))_{[1\to n, 1\to n_{cur}, true\to U]}, U), n_{cur})$$

Здесь $\mathbb{TP}$ – оператор трансляции пути p в промежуточный язык, $t(c)$ – таблица класса c. В общем случае $E_{[n\to n', n_{cur}\to n'_{cur}, U\to U']}$ означает, что вычисление выражения E переводит значение n в n', а n_{cur} в n'_{cur}, а условие U в условие U'. В операторе $\mathbb{TP}(p, E)_{[n\to n', n_{cur}\to n'_{cur}, U\to U']}$: E – выражение на промежуточном языке, порожденное на предыдущих этапах трансляции, n – текущий размер таблицы, n_{cur} – номер столбца, содержащий значение шага, продолжением которого является путь p, U' – условие на строки таблицы, сгенерированное в процессе исполнения оператора трансляции $\mathbb{TP}$ на основе входного условия U.

Для пустого p определяем:

$$\mathbb{TP}(\emptyset, E)_{[n\to n, n_{cur}\to n_{cur}, U\to U]} = E_{[n\to n, n_{cur}\to n_{cur}, U\to U]}$$

В частности,

$$\mathbb{T}(c()) = Select(Where(\mathbb{TP}(\emptyset, t(c))_{[1\to 1, 1\to 1, true\to true]}, true), 1)$$

Поскольку условие $true$ истинно на всех строках таблицы, то значением тривиального запроса $c()$ будет множество идентификаторов всех объектов класса c: $\{id \mid id \in Select(t(c), 1)\}$.

Определим оператор трансляции $\mathbb{TP}(p, E)$ для произвольного пути p. Сначала рассмотрим случай, когда левый шаг пути — это поле f и p $= f.p_1$ для некоторого пути p_1:

$$\mathbb{TP}(f.p_1, E)_{[n\to n\to n', n_{cur}\to n'_{cur}, U\to U']} =$$
$$\mathbb{TP}(p_1, Join(E, n_{cur}, t(f))_{[n\to n+1, n_{cur}\to n+1, U\to U]})_{[n+1\to n\to n', n+1\to n'_{cur}, U\to U']}$$

Заметим, что $Join$ не изменяет условие U. В частности, если p_1 – пустой, то

$$\mathbb{TP}(f, E)_{[n\to n+1, n_{cur}\to n+1, U\to U]} = Join(E, n_{cur}, t(f))_{[n\to n+1, n_{cur}\to n+1, U\to U]}$$

В случае $p = where(u).p_1$ имеем:

$$\mathbb{TP}(where(u).p_1, E)_{[n\to n', n_{cur}\to n'_{cur}, U\to U']} =$$
$$\mathbb{TP}(p_1, \mathbb{TC}(u, E)_{[n\to n'', n_{cur}\to n''_{cur}, true\to u^{\mathbb{TC}}]})_{[n''\to n', n''_{cur}\to n'_{cur}, (U\wedge u^{\mathbb{TC}})\to U']}$$

где $\mathbb{TC}$ — оператор трансляции условий, который определим ниже, а n'' и n''_{cur} — значения, в которые переходят n и n_{cur} соответственно после исполнения $\mathbb{TC}(u, E))$. u^{TC} — представление условия u на промежуточном языке, сгенерированное в результате применения оператора $\mathbb{TC}$. Это условие добавляется к условию U. Если p_1 пустой, то

$$\mathbb{TP}(where(u), E)_{[n\to n', n_{cur}\to n'_{cur}, U\to U\wedge u^{\mathbb{TC}}]} =$$
$$= \mathbb{TC}(u, E)_{[n\to n', n_{cur}\to n'_{cur}, true\to u^{\mathbb{TC}}]})$$

Определим теперь действие оператора $\mathbb{TC}$ и процедуру получения $u^{\mathbb{TC}}$. Сделаем это индукцией по структуре u. $\mathbb{TC}$ разбирает условие u, добавляя в результирующую таблицу результаты вычисления выражений из u с параллельным формированием $u^{\mathbb{TC}}$.

1) $u = p_1 \star p_2,\ \star \in \{=, \neq, >, <, \geq, \leq, \in\}$. Тогда

$$\mathbb{TC}(p_1 \star p_2, E)_{[n\to n\to n', n_{cur}\to n'_{cur}, true\to u^{\mathbb{TC}}]} =$$
$$\mathbb{TP}\big(p_2, \mathbb{TP}(p_1, E)_{[n\to n\to n'', n_{cur}\to n''_{cur}, true\to true]}\big)_{[n''\to n\to n', n''_{cur}\to n'_{cur}, true\to true]}$$

Здесь n'' и n''_{cur} – номера столбцов, полученные после применения $\mathbb{TP}$ к p_1. Для всего $\mathbb{TC}(p_1 \star p_2, E)$ определяем номера столбцов n' и n'_{cur} как равные номерам столбцов, полученным в результате применения оператора $\mathbb{TP}$ к p_2. Условие справа от равенства не формируется, поскольку p_1 и p_2 не содержат вхождения шага $where$, но сам $\mathbb{TC}$ генерирует условие $u^{\mathbb{TC}}$, которое для $p_1 \star p_2$ определим как

$$u^{\mathbb{TC}} = column(s, n''_{cur}) \star column(s, n'_{cur}).$$

2) $u = A \odot B$, где $\odot \in \{\wedge, \vee\}$. Тогда

$$\mathbb{TC}(A \odot B, E)_{[n \to n \to n', n_{cur} \to n'_{cur}, true \to u_A^{\mathbb{TC}} \odot u_B^{\mathbb{TC}}]} =$$

$$\mathbb{TC}(B, \mathbb{TC}(A, E)_{[n \to n \to n'', n_{cur} \to n''_{cur}, true \to u_A^{\mathbb{TC}}]})_{[n'' \to n \to n', n''_{cur} \to n'_{cur}, true \to u_B^{\mathbb{TC}}]},$$

где n'' и n''_{cur} – номера столбцов, полученные после применения $\mathbb{TC}$ к A. Номера столбцов для всего $\mathbb{TC}(A \odot B, E)$ равны номерам столбцов, полученным в результате применения $\mathbb{TC}$ к B. $u_A^{\mathbb{TC}}$ и $u_B^{\mathbb{TC}}$ – условия, сформированные при трансляции A и B соответственно.

3) $u = \neg A$. Тогда

$$\mathbb{TC}(\neg A, E)_{[n \to n \to n', n_{cur} \to n'_{cur}, true \to \neg u_A^{\mathbb{TC}}]} =$$

$$\mathbb{TC}(A, E)_{[n \to n \to n'', n_{cur} \to n''_{cur}, true \to u_A^{\mathbb{TC}}]}$$

где $u_A^{\mathbb{TC}}$ – условие для A.

Определение оператора $T(Q)$, переводящего запрос Q в промежуточный язык, завершено.

В следующем разделе показано, как термы промежуточного языка отображаются в SQL-код. На основе этого подхода на платформе Ontobox реализован компилятор языка запросов, который активно используется при решении практических задач. На настоящий момент корректность проверена практически – на основе набора тестовых примеров. Однако корректность на уровне промежуточного языка можно обосновать теоретически – через доказательство коммутативности следующей диаграммы:

$$\begin{array}{ccc} Q & \xrightarrow{\texttt{Eval}_{(\emptyset, Q)}} & \{o_1, \ldots, o_k\} \\ {\scriptstyle T}\downarrow & & \downarrow{\scriptstyle \nu} \\ T(Q) & \xrightarrow[\texttt{Eval}_{R(T(Q))}]{} & \{\nu(o_1), \ldots, \nu(o_k)\} \end{array} \tag{5.1}$$

Эта задача включена в план дальнейших исследований.

6. Компиляция в SQL

При переводе в SQL операции промежуточного языка играют роль макросов с той особенностью, что для уточнения семантики и оптимизации кода в процессе перевода активно задействуется информация из метаописания модели. Затем получившиеся в результате блоки подставляются в общий шаблон запроса (в синтаксисе Libretto):

```
SELECT #{resultBlock} FROM #{fromBlock}
  WHERE #{conditionBlock}
    #{if(orderBy) #" ORDER BY #{orderByBlock}" }
```

Все поддерживаемые на сегодняшний день запросы реализуются посредством описанной выше структуры следующим образом:

1) На основании информации из первичной выборки строится цепочка join'ов, формирующая полное множество выборки, до применения фильтрующих условий.

2) Булево выражение U, сформированное в процессе исполнения оператора T, уже представляет собой синтаксическое дерево результирующего условия, но поля могут иметь разный тип и в зависимости от того, какое поле и какое условие указаны в конкретной структуре, выбирается тот или иной вариант шаблона запроса. Структура, расположенная внутри данного поля, содержит конкретные значения, которые используются для сравнения со значениями полей документа, но на этапе формирования запроса вместо конкретных значений ставятся `?`, а сами значения выносятся в отдельный массив в нужном порядке.

3) После того как все составные компоненты запроса вычислены, они подставляются в основной шаблон, затем формируется объект подготовленного запроса (prepared statement), который необходим для того, чтобы безопасно подставлять пользовательские значения в запрос и обезопасить систему от различных известных уязвимостей вроде SQL-инъекций.

4) Данному объекту передаются конкретные значения из сформированного ранее массива, после чего выполняется сам запрос. По итогам запроса мы получаем последовательность кортежей, содержащих в себе всю промежуточную информацию, необходимую для запроса.

5) Из операции *Select* получается метаинформация о том, документы какого класса будут представлены в качестве результата данного запроса, а также информация о том, какой именно элемент из каждого из полученных кортежей необходимо интерпретировать в качестве результирующего документа. Элементы полученной в итоге последовательности идентификаторов оборачиваются в структуры `doc/Item` и возвращаются в качестве результата всего метода.

7. Апробация

В рамках апробации эффективность компилятора в SQL сравнивалась с предыдущей реализацией языка запросов, основанной на пошаговой интерпретации запросов в объектной модели. Результаты апробации продемонстрировали принципиальные преимущества компиляции относительно интерпретации. Измерения проводились на двух тестовых примерах. Первый пример – про внуков (3.1). Случайным образом были сгенерированы данные о 108 925 персонах, в результате возвращалось 63 должности, на которых работали люди, имеющие внуков младше 10 лет. Второй пример – поиск объектов, удовлетворяющих сложному арифметическому условию в пространстве 100 000 объектов. Результаты измерений в миллисекундах представлены в таблице:

Пример	Кол-во объектов	Интерпрет. без кеша	Интерпрет. кеш 1000	Компилятор без кеша	Компилятор кеш 1000
Тест 1	108 925	224 926 ms	21 055 ms	119 ms	65 ms
Тест 2	100 000	748 444 ms	28 552 ms	2 678 ms	117 ms

Поиск проводился как без использования кеша, так и с кешем с объемом в 1000 наиболее часто используемых объектов. В примерах компилятор дает ускорение — в 323 и 244 раза соответственно. Это означает, что только компиляция языка запросов делает возможным применение объектных моделей в качестве долговременных хранилищ данных при решении задач реальной сложности. Такое существенное различие объясняется большими накладными расходами на организацию пошаговых транзакций в случае интерпретатора. Эффективность скомпилированного объектного запроса сопоставима с эффективностью нативных SQL запросов, разработанных вручную.

8. Заключение

В работе исследуются вопросы реализации языка запросов к объектной модели как к долговременному хранилищу данных. В качестве языка запросов использовалось подмножество объектно ориентированного языка Libretto. Был разработан и реализован метод компиляции языка запросов в рамках low-code-платформы разработки приложений Ontobox. Продемонстрировано принципиальное улучшение эффективности исполнения компилируемого кода относительно интерпретируемого. В дальнейшем планируется как развитие самого языка запросов, так и продолжение теоретических исследований в этом направлении (в частности, доказательство корректности диаграммы (5.1)).

Список источников

1. Гаврилина Д. Э., Манцивода А. В. Low-code и объектные электронные таблицы // Известия Иркутского государственного университета. Серия Математика. 2022. Т. 40. С. 93–103. https://doi.org/10.26516/1997-7670. 2022.40.93
2. Малых А. А., Манцивода А. В. Документное моделирование // Известия Иркутского. госуниверситета. Серия Математика. 2017. Т. 21. С. 89–107. https://doi.org/10.26516/1997-7670.2017.21.89
3. Arul S. IT Professionals and DevOps Say No to Low-Code. AIM, January 2022. Available at: https://analyticsindiamag.com/it-professionals-and-devops-say-no-to-low-code/ (accessed 1 April 2022).
4. Business Process Model and Notation (BPMN). Version 2.0. Available at: https://www.omg.org/spec/BPMN/2.0/PDF (accessed 1 April 2022).
5. Gartner Review: Magic Quadrant for Enterprise Low-Code Application Platforms. 20 Sep 2021. Available at: https://www.gartner.com/doc/reprints?id=1-27I04ZJT&ct=210921&st=sb (accessed 1 April 2022).
6. Malykh A., Mantsivoda A. Query Language for Logic Architectures// Proceedings of 7th International Conference “Perspectives of System Informatics”. Springer-Verlag, 2010. (Lecture Notes in Computer Science ; 5947). P.294–305.
7. Mantsivoda A.V., Ponomaryov D.K. Towards Semantic Document Modelling of Business Processes // Известия Иркутского государственного университета. Серия Математика. 2019. Т. 29. С. 52-67. https://doi.org/10.26516/1997-7670.2019.29.52
8. Martin J., Gupta S., Kumar A. Scaling with Low Code to Accelerate Digital Transformation. HFS Research & Infosys. Published November 2021. Available at: https://www.infosys.com/services/digital-process-automation/insights/scaling-low-code.html (accessed 1 April 2022).
9. Wolff E. Microservices: Flexible Software Architecture. Addison-Wesley Professional, 2016. 432 p.
10. Wayner P. Why Developers Hate Low-Code. 9 Reasons Programmers Grow Frustrated with the Tools That are Supposed to Save Them Time // InfoWorld. 2019. Sep. 16. Available at: https://www.infoworld.com/article/3438819/why-developers-hate-low-code.html (accessed 1 April 2022).

References

1. Gavrilina D.E., Mantsivoda A.V. Low-code and Object Spreadsheats. it The Bulletin of Irkutsk State University. Series Mathematics, 2022. vol. 40, pp. 93–103. (in Russian) https://doi.org/10.26516/1997-7670.2022.40.93
2. Malykh A.A., Mantsivoda A.V. Document Models. *The Bulletin of Irkutsk State University. Series Mathematics*, 2017, vol. 21, pp. 89–107. (in Russian) https://doi.org/10.26516/1997-7670.2017.21.89
3. Arul S. IT Professionals and DevOps Say No to Low-Code. AIM, January, 2022. Available at: https://analyticsindiamag.com/it-professionals-and-devops-say-no-to-low-code/ (accessed 1 April 2022).
4. Business Process Model and Notation (BPMN). Version 2.0. Available at: https://www.omg.org/spec/BPMN/2.0/PDF (accessed 1 April 2022).
5. Gartner Review: Magic Quadrant for Enterprise Low-Code Application Platforms. 20 Sep 2021. Available at: https://www.gartner.com/doc/reprints?id=1-27I04ZJT&ct=210921&st=sb (accessed 1 April 2022).

6. Malykh A., Mantsivoda A. Query Language for Logic Architectures. *Proceedings of 7th International Conference 'Perspectives of System Informatics'*, Springer-Verlag, Lecture Notes in Computer Science 5947, 2010, pp. 294–305.
7. Mantsivoda A.V., Ponomaryov D.K. Towards Semantic Document Modelling of Business Processes. *The Bulletin of Irkutsk state university. Series Mathematics*, 2019, vol. 29, pp. 52–67. https://doi.org/10.26516/1997-7670.2019.29.52
8. Martin J., Gupta S., Kumar A. Scaling with Low Code to Accelerate Digital Transformation. HFS Research & Infosys. Published November 2021. Available at: https://www.infosys.com/services/digital-process-automation/insights/scaling-low-code.html (accessed 1 April 2022).
9. Wolff E. *Microservices: Flexible Software Architecture*. Addison-Wesley Professional, 2016, 432 p.
10. Wayner P. Why Developers Hate Low-Code. 9 Reasons Programmers Grow Frustrated with the Tools That are Supposed to Save Them Time // InfoWorld, Sep 16, 2019. Available at: https://www.infoworld.com/article/3438819/why-developers-hate-low-code.html (accessed 1 April 2022).

Об авторах

Гаврилин Денис Николаевич, аспирант, Иркутский государственный университет, Российская Федерация, 664003, г. Иркутск

Кустова Ирина Александровна, ведущий разработчик, ООО «Логус», Российская Федерация, 664003, г. Иркутск

Манцивода Андрей Валерьевич, д-р физ.-мат. наук, проф., Иркутский государственный университет, Российская Федерация, 664003, г. Иркутск

About the authors

Denis N. Gavrilin, Postgraduate, Irkutsk State University, Irkutsk, 664003, Russian Federation

Irina A. Kustova, Principal Developer, Logus Ltd., Irkutsk, 664003, Russian Federation

Andrei V. Mantsivoda, Dr. Sci. (Phys.Math.), Prof., Irkutsk State University, Irkutsk, 664003, Russian Federation

Поступила в редакцию / Received 12.02.2022
Поступила после рецензирования / Revised 12.09.2022
Принята к публикации / Accepted 26.09.2022