Сквозная функциональность и её анализ в грамматике языка программирования

Алексей Головешкин

Южный федеральный университет
Институт математики, механики и компьютерных наук
им. И. И. Воровича

Аспект (Aspect)

единица модульности, реализующая некоторую сквозную функциональность — действия, которые должны быть выполнены в разных местах программы, при этом, как правило, не относящиеся напрямую к основной логике (к логике предметной области)

Приоритетная декомпозиция (Dominant decomposition)

разбиение программы на единицы модульности, зафиксированное на этапе проектирования

Рис. 1: Классические примеры аспектов (Л – логирование, Д – проверка прав доступа)

Функциональность (Concern)

проблема, решаемая совокупностью фрагментов кода, и сама данная совокупность

Сквозная функциональность (Crosscutting concern)

функциональность, фрагменты кода которой рассредоточены по разным файлам, классам, методам проекта

Определение прорезающего функционала **не содержит** предположений о возможности вынесения данного функционала в отдельную единицу модульности.
Ключевая идея АОП — идея разделения кода аспекта и кода основной программы[1].

[1] Masuhara H., Kiczales G., Modeling Crosscutting in Aspect-Oriented Mechanisms. // Object-Oriented Programming, ser. LNCS. — Springer-Verlag, 2003. — Vol. 2743. — С. 2–28.

А. Л. Фуксман, "Технологические аспекты создания программных систем"

- приоритетная декомпозиция $\sim$ иерархически организованный набор реализующих функций $\sim$ горизонтальные слои программы
- сквозные функциональности $\sim$ вертикальные слои $\sim$ расширяющие функции, прорезающие приоритетную декомпозицию
- для изучения сквозных функциональностей в интегрированной программе необходимо наличие их **сосредоточенного описания** или **послойная разметка** программы

- Компилятор — программа, содержащая сквозную функциональность, непригодную для представления в виде аспекта.
- Новая языковая конструкция — функционал, прорезающий фазы компиляции, составляющие приоритетную декомпозицию.
- Оформление языковых конструкций в виде аспектов (если оно возможно) добавляет новый уровень декомпозиции и усложняет восприятие разрабатываемого языка как единого целого.
- Необходимы разметка функциональностей и анализ их взаимосвязей в интегрированной программе.

- Грамматика ЯП — структурированный текст, принципиально отличающийся от объектно-ориентированной программы.
- Разметка функциональностей в грамматике языка предполагает выделение правил, относящихся к реализации той или иной языковой конструкции.
- Информация о зависимостях позволяет оценить безопасность модификации и отключения функциональностей языка.

S — множество различных функциональностей, которые могут быть выделены в программе.
T — множество, элементов программы для некоторой степени детализации.

Определяются[2] отображение $f : S \to 2^T$, связывающее каждую функциональность $s \in S$ с множеством реализующих её элементов программы, и отображение $g : T \to 2^S$, связывающее элемент со всеми функциональностями, в реализации которых он участвует.

[2] Conejero J. M., Hernández J., Jurado E., Berg K. G. van den, Crosscutting, what is and what is not?: A Formal definition based on a Crosscutting Pattern. // University of Twente. URL: http://doc.utwente.nl/64648/1/ConHerJurBer2007.pdf

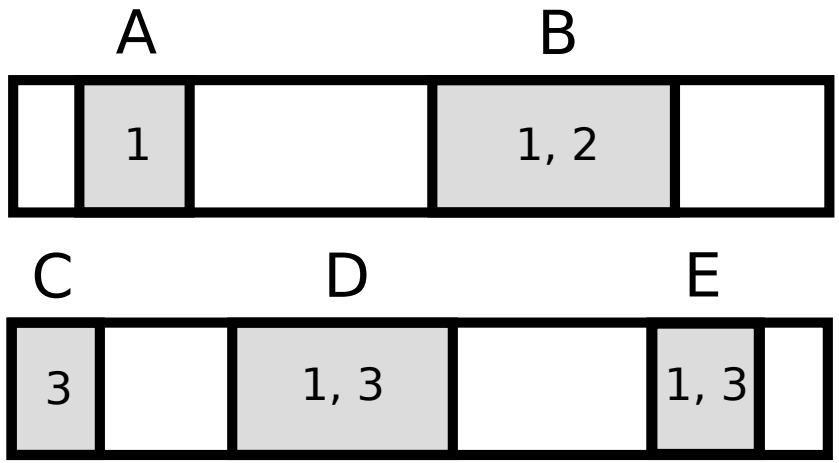

Рис. 2: Пример прорезающих функциональностей

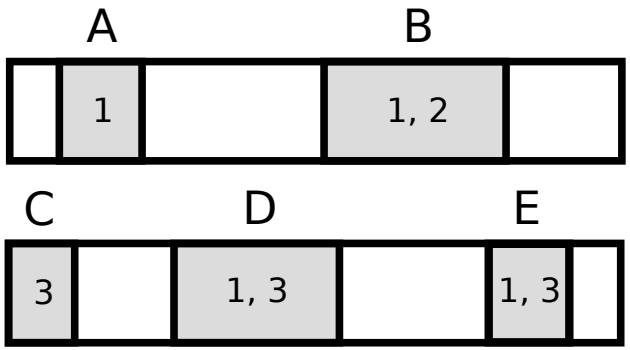

Рис. 2: Пример прорезающих функциональностей

$$S = \{1, 2, 3\}$$
$$T = \{A, B, C, D, E\}$$

$$f(1) = \{A, B, D, E\}$$
$$f(2) = \{B\}$$
$$f(3) = \{C, D, E\}$$

$$g(A) = \{1\}$$
$$g(B) = \{1, 2\}$$
$$g(C) = \{3\}$$
$$g(D) = \{1, 3\}$$
$$g(E) = \{1, 3\}$$

Прорезание (Crosscutting) = Распределённость (Scattering) + Спутанность (Tangling)

Распределённая функциональность реализуется несколькими элементами. Спутанный элемент реализует несколько функциональностей.

Прорезание как сочетание распределённости и спутанности для пары функциональностей $s_1, s_2 \in S : s_1 \neq s_2$ имеет место тогда и только тогда, когда

$$|f(s_1)| > 1\,,$$
$$\exists t \in f(s_1) : s_2 \in g(t).$$

- интуитивно: прорезание означает прорезание функциональностью всей программы;
- Conejero, et al.: прорезание означает прорезание одной функциональностью другой функциональности.

Рис. 1: Классические примеры аспектов (Л – логирование, Д – проверка прав доступа)

Типы связей между функциональностями и элементами программы

- непосредственная связь: элемент программы реализует логику, составляющую функциональность
- опосредованная связь: участок A не связан с функциональностью 1 непосредственно, но существует участок A', непосредственно связанный с 1 и некоторым образом зависящий от A

```
if_stmt: IF expr THEN ...
       ;

while_stmt: WHILE expr DO ...
          ;
```

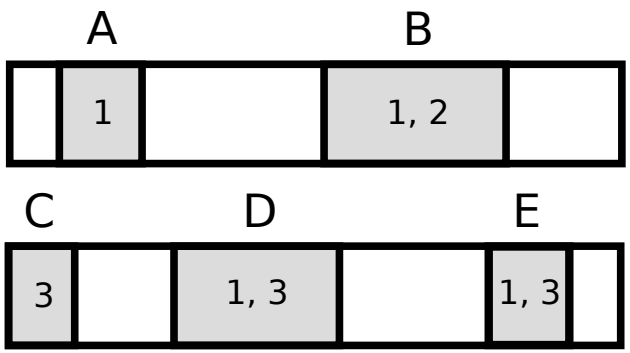

Рис. 2: Пример прорезающих функциональностей

	A	B	C	D	E
1	1	1	0	1	1
2	0	1	0	0	0
3	0	0	1	1	1

Таблица 1: Матрица зависимостей (dM)

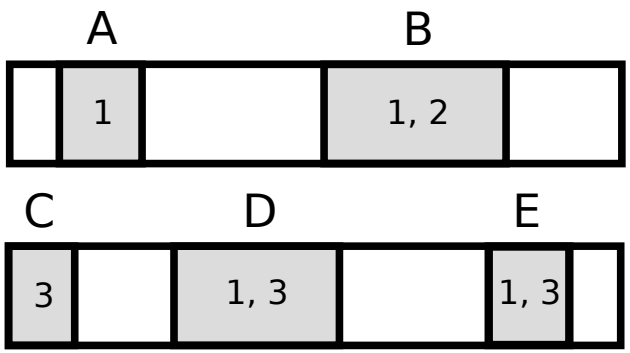

Рис. 2: Пример прорезающих функциональностей

	1	2	3
1	3	1	2
2	0	0	0
3	2	0	2

Таблица 2: Матрица сквозного функционала (ccM)

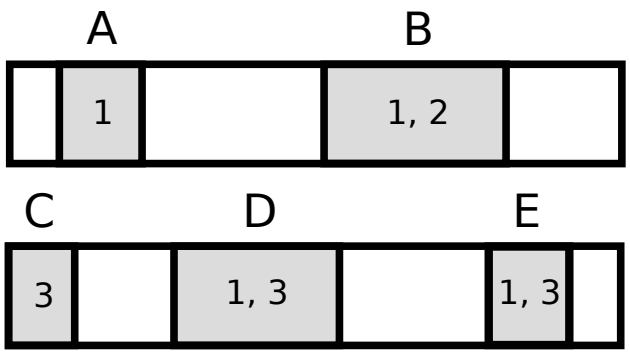

Рис. 2: Пример прорезающих функциональностей

	1	2	3
1	0	1	1
2	0	0	0
3	1	0	0

Таблица 3: Матрица сквозного функционала (ccM')

В качестве абсолютной величины, позволяющей оценить масштаб прорезания функциональностью s_i других функциональностей системы, предлагается[3] метрика *Степень прорезания* (*СП*):

$$СП(s_i) = \frac{ccM_{ii} + \sum_{j=1}^{|S|} ccM'_{ij}}{|S| + |T|}.$$

[3] Conejero J. M., Figueiredo E., Garcia A., Hernández J., Jurado E., Early Crosscutting Metrics as Predictors of Software Instability. // Objects, Components, Models and Patterns, ser. LNBIP. — Springer Berlin Heidelberg, 2009. — Vol. 33. — C. 136–156

- Предмет анализа - грамматика языка PascalABC.NET.
- Базовый инструмент - интегрированная среда YACC MC с поддержкой теговой разметки грамматики[4].
- Разработчиками PascalABC.NET размечено 20 различных функциональностей.

[4] Головешкин А. В., IDE с аспектной разметкой кода для работы с YACC-грамматиками. // Магистерская диссертация. — Южный федеральный университет, Ростов-на-Дону, 2015. — 53 с.

Функциональность	Кол-во	Функциональность	Кол-во
Выражения	32	*Типы*	17
Списки	50	*Ключевые слова*	6
Знаки операций	10	*ShortFuncDefinition*	3
Конст. выражения	20	*new_expr*	3
Константы	15	*TemplateTypeName*	3
Описания	24	*FuncName*	4
Операторы	20	*Проблема с атрибутами*	6
Имена	24	*Лямбды*	15
Секции	19	*Кортежи*	2
Заголовки	11	*Элементы списка*	24

Таблица 4: Функциональности грамматики PascalABC.NET и количество помеченных символов

Метрика *СП* естественным образом адаптирована для анализа попарной степени прорезания двух распределённых функциональностей:

$$\textit{Попарная степень прорезания}(s_i, s_j) = \frac{ccM_{ij}}{|f(s_i) \cup f(s_j)|}\ .$$

Этапы анализа

- Первый этап: учёт непосредственно связанных с функциональностями символов грамматики (тегированных символов).
- Второй этап: учёт опосредованных связей (учёт символов, определяющих непосредственно связанные элементы).

Списки

```
...
case Month of
  1,2,12: Season := 'Зима';
  ...
end;
...
except
  on System.DivideByZeroException do
    writeln('Целочисленное деление на 0');
  on System.IO.IOException do
    writeln('Файл отсутствует');
end;
```

Лямбды

```
(x,y) -> x*y
(x,y: integer) -> x*y
(x,y: integer): integer -> x*y
(x: integer; y: integer) -> x*y
```

Кортежи

```
var t: (string, integer);
t := ('Иванов', 23);
```

	H	O	$H \cup O$
Списки	50	84	96
Кортежи	2	15	17
Лямбды	15	49	52

Таблица 5: Количество элементов, непосредственно и опосредованно связанных с конструкциями языка

s_1, s_2	$H(s_1) \cap H(s_2)$	$O(s_1) \cap O(s_2)$	$(H(s_1) \cup O(s_1)) \cap (H(s_2) \cup O(s_2))$
Списки, *Кортежи*	0 (0)	0.0421 (4)	0.0367 (4)
Лямбды, *Кортежи*	0 (0)	0.1636 (9)	0.1897 (11)
Списки, *Лямбды*	0.0156 (1)	0.0726 (9)	0.0725 (10)

Таблица 6: *ПСП*(s_1, s_2) и количество спутанных элементов

- Если функциональности пересекаются своими непосредственно связанными элементами, ни одну нельзя исключить без нарушения другой.
- Иначе в ситуации, когда пересечение возникает при рассмотрении опосредованно связанных элементов, возможны варианты:
  - имеет место спутанность только на уровне сервисных функций (вспомогательных символов), не относящихся к конкретной функциональности;
  - элемент, непосредственно связанный с одной функциональностью, опосредованно связан с другой.

Несимметричная оценка, не учитывающая зависимость на уровне сервисных функций:

$$\textit{Степень зависимости}(s_i, s_j) = \frac{|H(s_i) \cap (H(s_j) \cup O(s_j))|}{|H(s_i)|},$$

где $H(s_i)$, $H(s_j)$ — множества элементов, непосредственно связанных с соответствующими функциональностями, $O(s_i)$, $O(s_j)$ — множества опосредованно связанных элементов.

	H	O	$H \cup O$
Списки	50	84	96
Кортежи	2	15	17
Лямбды	15	49	52

Таблица 5: Количество элементов, непосредственно и опосредованно связанных с конструкциями языка

s_1, s_2	$СЗ(s_1, s_2)$	s_1, s_2	$СЗ(s_1, s_2)$
Списки, *Кортежи*	0	*Кортежи*, *Списки*	0
Лямбды, *Кортежи*	0.2	*Кортежи*, *Лямбды*	0.5
Списки, *Лямбды*	0.04	*Лямбды*, *Списки*	0.27

Таблица 7: Значения степеней зависимости для рассматриваемых функциональностей

- Применение обобщённой формальной модели к размеченной программе позволяет выделить прорезающие функциональности даже в случае неприменимости классического АОП.
- При оценке зависимости двух функциональностей важно учитывать не только непосредственно относящиеся к ним элементы программы, но и опосредованно связанные.
- Количественная оценка зависимостей и их качественный анализ предоставляют информацию, которую необходимо учитывать при модификациях функциональностей языка.
- Обнаружение зависимости между функциональностями грамматики и её количественная оценка позволяют высказать предположение о зависимости этих функциональностей в самом компиляторе как объектно-ориентированной программе.

- Kienzle J., Yu Y., Xiong J., On Composition and Reuse of Aspects. // University of Central Florida. URL: http://www.eecs.ucf.edu/leavens/FOAL/papers-2003/kienzle-yu-xiong.pdf
- Kaindl H., What is an Aspect in Aspect-oriented Requirements Engineering. // Proceedings of EMMSAD. — 2008. — Vol. 337 — C. 164–170.
- Masuhara H., Kiczales G., Modeling Crosscutting in Aspect-Oriented Mechanisms. // Object-Oriented Programming, ser. LNCS. — Springer-Verlag, 2003. — Vol. 2743. — C. 2–28.
- Фуксман А. Л., Технологические аспекты создания программных систем. / А. Л. Фуксман — М: Статистика, 1979. — 184 с.
- Горбунов-Посадов М. М., Как растёт программа. // ИПМ им. М. В. Келдыша РАН. URL: http://keldysh.ru/gorbunov/grow.htm
- Conejero J. M., Hernández J., Jurado E., Berg K. G. van den, Crosscutting, what is and what is not?: A Formal definition based on a Crosscutting Pattern. // University of Twente. URL: http://doc.utwente.nl/64648/1/ConHerJurBer2007.pdf
- Berg K. G. van den, Conejero, J. M., Hernández J., Identification of Crosscutting in Software Design. // 8th International Workshop on Aspect-Oriented Modeling. — University of Duisburg-Essen, 2006. — C. 1–7.
- Conejero J. M., Figueiredo E., Garcia A., Hernández J., Jurado E., Early Crosscutting Metrics as Predictors of Software Instability. // Objects, Components, Models and Patterns, ser. LNBIP. — Springer Berlin Heidelberg, 2009. — Vol. 33. — C. 136–156.
- Головешкин А. В., IDE с аспектной разметкой кода для работы с YACC-грамматиками. // Магистерская диссертация. — Южный федеральный университет, Ростов-на-Дону, 2015. — 53 с.