ПРИМЕНЕНИЕ ДЕСКРИПТОРОВ ПРИ НАПИСАНИИ ТРАНСПАЙЛЕРА НА ПРИМЕРЕ ТРАНСПАЙЛЕРА ИЗ C# В DART

Ващилко А. П.
Кафедра информатики, Белорусский государственный университет информатики и радиоэлектороники
Минск, Республика Беларусь
E-mail: alexander.armoken@gmail.com

Эта статья может помочь тем, кто пишет конвертер из одного высокоуровнего языка программирование в другой и решил реализовать правила конвертации, например, для конвертации вызовов стандартных библиотек. В качестве примеров таких языков в статье используются C# и Dart.

Введение

При написании конвертора одного высокоуровнего языка программирования в другой, может возникнуть необходимость интегрировать сконвертированный код в уже существующую кодовую базу. Чтобы это реализовать необходимо указать конвертору какие синтаксические или семантические конструкции нужно сконвертировать таким образом, чтобы их можно было без дополнительных ручных правок сразу же использовать. Для этого необходимо реализовать правила трансляции.

I. Правила трансляции

Правило трансляции состоит из синтаксического и семантического описания пары языковых конструкций:

1. входной конструкции на языке C# (является ключом, по которому ищется правило);
2. образец конструкции на языке Dart получаемой на выходе программы.

Поиск правила осуществляется последовательным сравнением языковой конструкции из конвертируемого кода и языковой конструкции, описываемой ключом правила.

Наиболее гибким является использование деревьев (не обязательно синтаксических, так как их проблематично использовать, например, для описания типов) для описания языковых конструкций.

Во время реализации конвертора был использован компилятор с открытым исходным кодом Roslyn [1], поскольку он берет на себя работу по синтаксическому и семантическому анализу кода. Однако синтаксические деревья и графы, описывающие типы и члены типов, предоставляемые Roslyn, хотя и позволяют их сравнивать, но не дают возможности делать следующие вещи:

- ограничивать глубину сравнения;
- делать метки на узлах дерева;
- хранить синтаксическую и семантическую информации в одной структуре данных.

Чтобы не реализовывать весь функционал Roslyn можно реализовать обертки над его синтаксическими деревьями и графами, описывающими типы и их члены. Эти обертки в дальнейшем будут называться дескрипторами.

II. Дескрипторы

Дескрипторы представляют собой реализацию красно-зеленых деревьев [2] в случае синтаксических деревьев и неизменяемый фасад над изменяемыми данными в случае графов, описывающих тип и их члены, так как, например, при использовании шаблонов (CRTP [3]) возможны циклы в графе.

Дескрипторы необходимо использовать только для тех синтаксических конструкций, для интерпретации которых необходима масса семантической информации, для остальных можно использовать адаптеры над конструкциями Roslyn (RoslynExprAdapter). Например, выражения, описывающие цикл for в простых случаях не требует реализации отдельного условия, но бинарное выражение, описывающее сравнение двух переменных, уже требует этого. Данный подход позволяет реализовывать функционал конвертора по мере возникаемых требований.

Для остановки сравнения деревьев стоит использовать затычки (StubDescr), которые при сравнении с любым дескриптом из иерархии своего языка являются эквивалентными ему. Два дескриптора являются эквилентными не только тогда, когда все их узлы содержат одинаковую информацию о коде, но и когда некоторые из узлов являются затычками.

На все дескрипторы можно ставить текстовые метки, которые должны быть одинаковы в двух частях правила трансляции. Метки можно сделать единственными изменяемыми полями в общем-то неизменяемых дескрипторах.

Как уже можно было понять, для каждого языка необходима своя иерархия дескрипторов, даже если они будут идентичны по своим свойствам и функциям. Это предотвратит будущие архитектурные ошибки и упростит рефакторинг.

III. Применение правила трансляции

Для применения правил трансляции необходимо произвести следующие операции:

1. построить дескриптор языковой конструкции;
2. последовательно сравнить полученный дескриптор с ключами правил трансляции, пока не будет найдено подходящее правило (см. рисунки 1 и 2);
3. скопировать метки из дескриптора ключа в только что построенный дескриптор (см. рис. 2);
4. используя метки и образец выходной конструкции из правила, построить выходную конструкцию (см. рис. 3). Стоит уделить внимание тому, что метки в левой части рисунка 3 точно такие же как и в правой части рисунка 2. Отвечает за консистентность меток разработчик правил, но разработчик транспайлера может помочь ему, если реализует проверку консистентности меток во время запуска транспайлера;
5. обойдя граф выходной конструкции, получить строковое представление кода. Эта операция не является обязательной, если планируется использование дескриптора Dart для каких-либо иных манипуляций над кодом.

Заключение

Использование дескрипторов позволяет спроектировать гибкую архитектуру транспайлера с учетом будущих потребностей и ограниченных ресурсов в настоящем. Поскольку необязательно реализовывать дескриптор для каждого вида языковой конструкции, а только лишь для тех, что необходимы в данный момент. Особенно эффективного использования дескрипторов можно достичь при создании узкоспециализированных инструментов разработчиков, если уже есть готовые библиотеки для анализа программного кода.

Список литературы

1. Компилятор C# Roslyn [Электронный ресурс] – Режим доступа: https://github.com/dotnet/roslyn. – Дата доступа: 10.10.2019.
2. Persistence, Facades and Roslyn's Red-Green Trees [Электронный ресурс] / Eric Lippert – Режим доступа: https://blogs.msdn.microsoft.com/ericlippert/. – Дата доступа: 10.10.2019.
3. C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond /David Abrahams, Aleksey Gurtovoy. Издательство: Addison-Wesley Professional, 2004. – 390с.

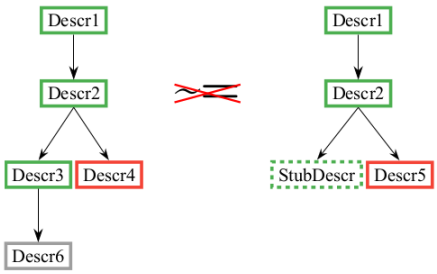

Рис. 1 – Пример неудачного сравнения дескрипторов

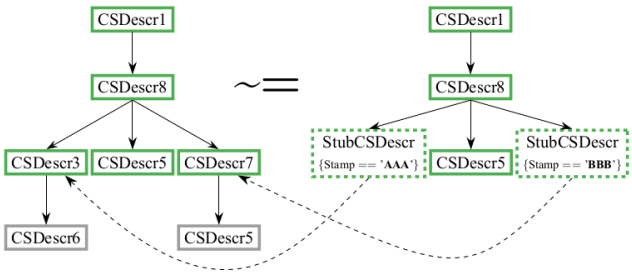

Рис. 2 – Пример удачного сравнения дескрипторов и копирования меток

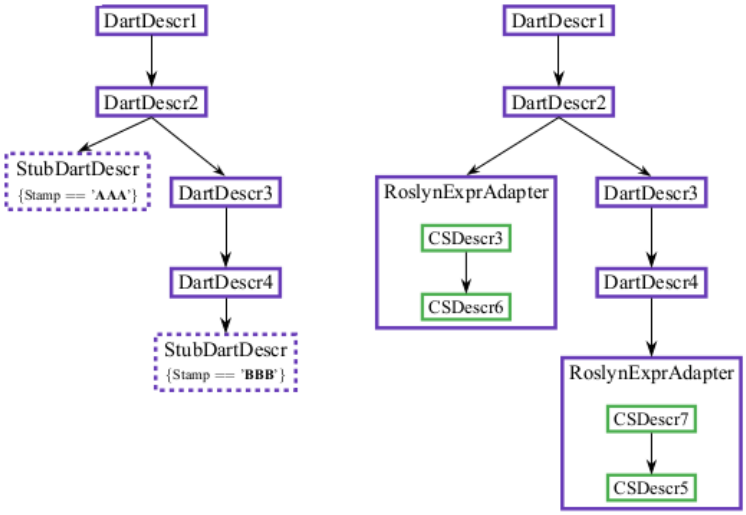

Рис. 3 – Пример вставки отмеченных дескрипторов