УДК 004.4'242

Бузовский О.В., д.т.н.,
Алещенко А.В.

СИСТЕМА ВЕРИФИКАЦИИ ГРАФИЧЕСКИХ СХЕМ АЛГОРИТМОВ И ГЕНЕРАЦИИ ПРОГРАММНЫХ КОДОВ

Национальный технический университет Украины "КПИ"
alexey.aleshchenko@gmail.com

Рассмотрены возможности графической нотации схем алгоритмов, проблемы описания типов и объявления переменных (типизация) в случае использования ГСА или UML-диаграммы деятельности, а также способы их верификации и трансляции. Приведено собственное решение данной задачи и его реализация в разрабатываемой системе

Ключевые слова: графические схемы алгоритмов, *UML*, типизация, верификация, трансляция, генерация программных кодов

Введение

В настоящее время существует большое количество *CASE*-систем (*Computer-Aided System/Software Engineering*), которые позволяют автоматизировать процесс разработки программных продуктов (ПП), повышая тем самым эффективность самого процесса. При этом подавляющее большинство *CASE*-систем ориентированы на применение унифицированного языка моделирования *UML* (*Unified Modeling Language*). Этот язык является основным инструментом *CASE*-технологий при объектно-ориентированном подходе к разработке ПП и включает в себя систему различных диаграмм, на основании которых может быть построено представление о проектируемом продукте.

Постановка задачи

Примерами программных средств, реализующих описанный во введении подход, являются: «*Rational Software Modeler*», «*ARIS Toolset*», «*Borland Together*», «*Sybase PowerDesigner*», «*Enterprise Architect*» и другие [1, с. 181]. Эти средства позволяют документировать проект и построить набор диаграмм и спецификаций, целью которых является получение каркаса ПП (диаграммы классов) который описывает структуру классов, интерфейсов и отношения между ними. Эта информация достаточна для генерации кода каркаса с так называемыми «заглушками», определяющими реализацию конкретных методов, но не позволяет получить полный исполняемый код.

Следовательно, *CASE*-средства не позволяют полностью автоматизировать процесс создания ПП, хотя большой процент затрат в рамках проекта связан именно с кодированием тел методов. Разрабатываемая в данной тематике система призвана свести ручное кодирование к минимуму, а в конечной цели – полностью отказаться от него.

Цель исследования

Сокращение рутинных трудозатрат на кодирование в процессе разработки программного обеспечения за счёт генерации исполняемых программных кодов по графической нотации схем алгоритмов.

Типизация переменных

Одной из причин затруднения автоматического построения кода есть проблема, связанная с описанием типов и объявлением переменных [2, с. 18]. В процессе разработки системы рассматривались следующие варианты задания соответствия между переменными и их типами: именная, динамическая и явная. При этом учитывалось следующее:

– именная типизация означает, что в зависимости от данных, которые хранит в себе переменная, модифицируется её имя. Например, используется добавление к имени переменной символа «i» и т.п. [3, с. 66]. В результате нарушается семантика имён и затрудняется описание подтипов;

– динамическая типизация предполагает, что предварительно тип переменной не задаётся, а определяется по мере выполнения программы (динамически), соответственно значению присваивания. Так тип одной переменной может меняться в зависимости от значений, присваиваемых ему, и операций, в которых он принимает участие, что затрудняет как контроль типов и отладку программы, а также разработку транслятора;

– при явной типизации очевидные преимущества это: наглядность, надёжность и распространённость. Именно этот вариант типизации используется для задания соответствия между переменной и её типом в реализуемой системе.

Два варианта системы

Система представлена в двух вариантах. Первый вариант реализует структурную парадигму программирования, т.е. трансляцию кода структурной программы, заданной в виде граф-схемы алгоритма (ГСА). Кроме того, эта реализация используется в целях обучения структурной парадигме программирования. Второй вариант поддерживает объектно-ориентированную парадигму и выполняет трансляцию внутреннего кода методов классов, заданного *UML*-диаграммой деятельности (*activity diagram*).

Применительно к первой технологии важно отметить, что ГСА не содержит нотации, предусматривающей описание типов. Поскольку, при разработке системы, предпочтение отдаётся явной типизации, соответствие между переменной и типом задаётся пользователем, т.е. требуются дополнительная информация, помимо ГСА и диаграммы деятельности. Эта информация может быть представлена в текстовом виде (например, как в языке Паскаль раздел типов и раздел переменных) или таблично. Именно второй способ использован в разработанной системе (рис. 1). Таблица предлагается системой, а пользователю в неё необходимо ввести соответствующие данные.

Во втором случае, при использовании диаграммы деятельности, также не предусмотрены средства описания типа переменных, однако информация о типах может быть получена из диаграммы классов. Эта информация в общем случае не является достаточной для автоматизации кодирования, поскольку требует описания типов локальных переменных (параметры цикла, буферные переменные и т.п.). Для того чтобы объявить эти переменные, также необходимы средства, которые описывались выше (либо текстовые, либо табличные).

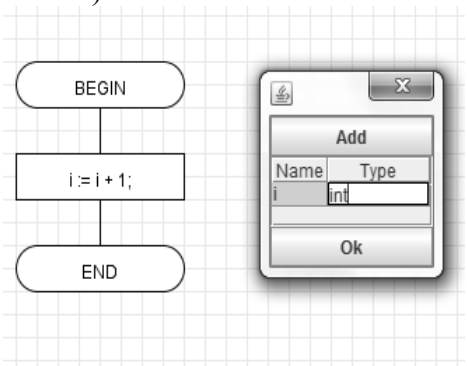


Рис. 1. Явная типизация в табличном виде

Верификация структуры ГСА

Для успешной генерации кода требуется предварительная проверка корректности задания самой ГСА.

Возможны следующие ошибки структуры ГСА:

– отсутствие терминальных вершин (начало и конец);

– множественность терминальных вершин одного класса;

– наличие бесконечных циклов (частично, в той мере, в какой оно определяется структурой блок-схемы);

– недостижимость вершины из начальной;

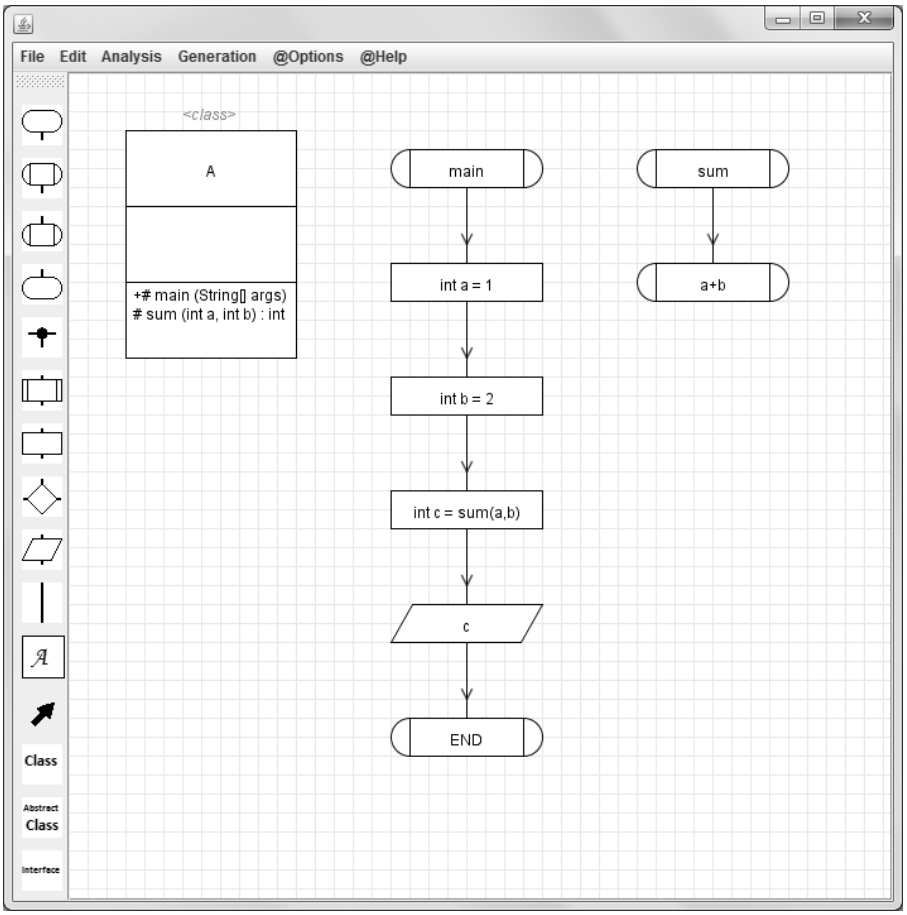
– наличие вершин, из которых отсутствует путь в конечную вершину.

Также возможным является выявление семантической ошибки ГСА, которая заключается в отсутствии изменений в теле цикла значений переменных входящих в состав условия завершения цикла.

Трансляция кода

Трансляция исходного кода заданного в графической форме с добавлением табличной типизации в исполняемый код

возможна тремя способами. Первый из них – использование промежуточного языка, второй – трансляция в унифицированный код (например, байт-код в языке *Java*), и третий – трансляция непосредственно в исполняемый код. Сравнение методов и обоснование выбора не являются предметом данной статьи. В реализуемой системе предпочтение отдано первому способу, где в качестве промежуточного языка выбран язык *Java*. Пример работы разрабатываемой системы приведен ниже (рис. 2).



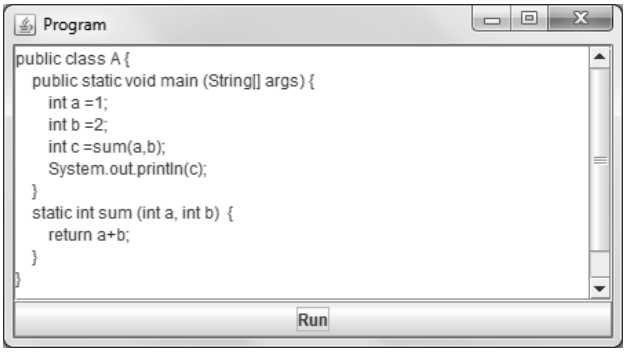


Рис. 2. Исходная ГСА в разрабатываемой системе и сгенерированный код на языке *Java*

Разрабатываемая система позволяет строить иерархию классов и интерфейсов (рис. 3), которая может быть дополнена описанием полей и методов.

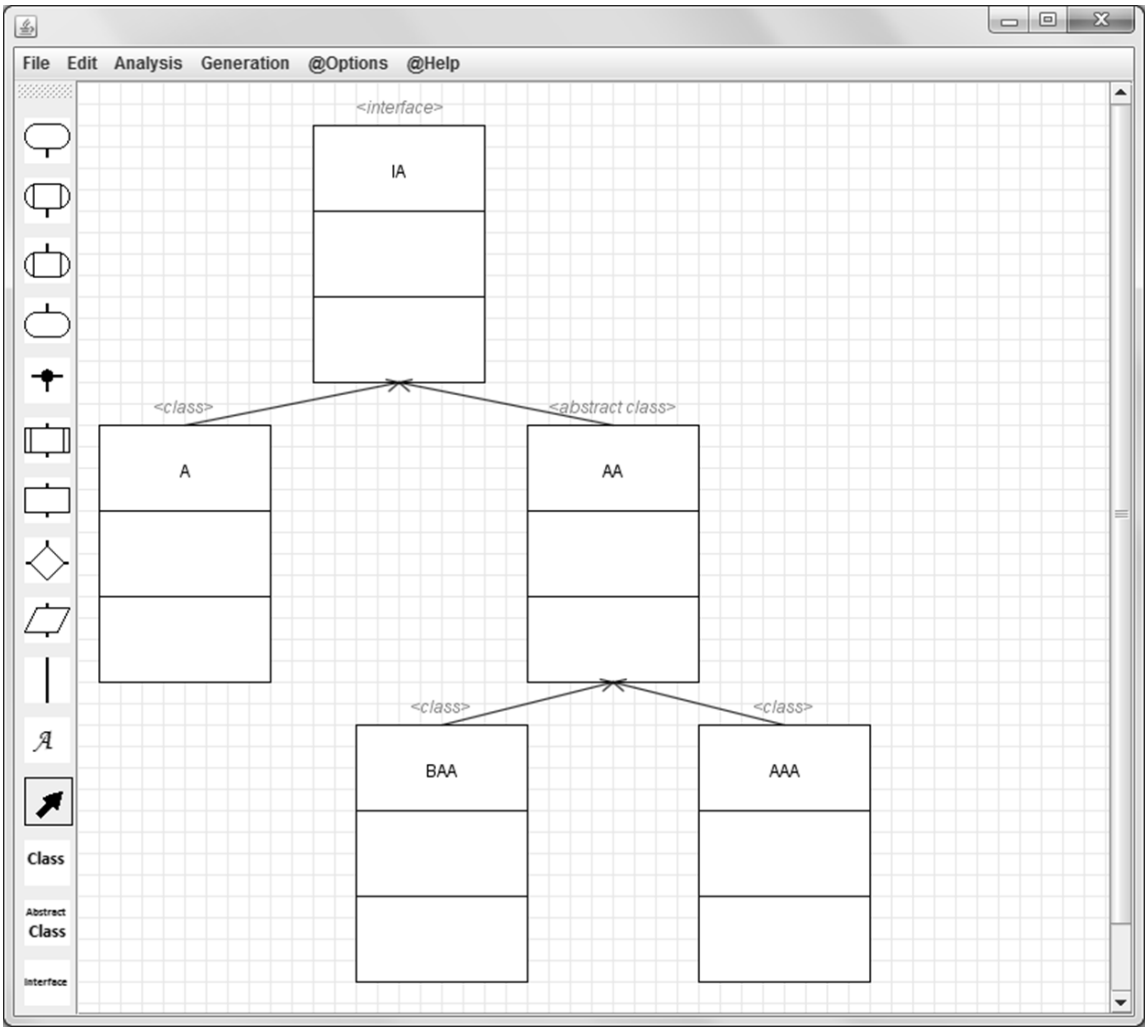


Рис. 3. Исходная диаграмма классов в разрабатываемой системе

Выводы

В статье проанализированы и оценены возможности графической нотации схем алгоритмов. Рассмотрены проблемы описания типов и объявления переменных (типизация) в случае использования ГСА или *UML*-диаграммы деятельности, а также способы их верификации и трансляции.

Приведено собственное решение данной задачи и его реализация в разрабатываемой системе для случая применения структурной и объектно-ориентированной парадигмы. Этот вариант системы может быть использован в целях обучения.

Список литературы

1. Гайнуллин Р. Ф. Разработка методов и средств анализа и контроля диаграмматики бизнес-процессов в проектировании автоматизированных систем: дис. кандидата технических наук : 05.13.12 / Гайнуллин Ринат Фаязович. – Ульяновск, 2014. – 189 с.

2. Вирт Н. Алгоритмы + Структуры данных = Программы / Вирт Н. – М. : Мир, 1985. – 406с.

3. Роберт У. Себеста. Основные концепции языков программирования / Роберт У. Себеста – М. : Издательский дом "Вильямс", 2001. – 672.

Статью представлено в редакцию 26.05.2015