АВТОМАТИЗАЦІЯ ТА УПРАВЛІННЯ МЕХАНІКО-ТЕХНОЛОГІЧНИМИ СИСТЕМАМИ ТА КОМПЛЕКСАМИ

УДК 004.4'242

О. В. БУЗОВСКИЙ, А. В. АЛЕЩЕНКО

СИСТЕМА ВИЗУАЛЬНОГО ПРОЕКТИРОВАНИЯ И ГЕНЕРАЦИИ ПРОГРАММНЫХ КОДОВ

Проанализированы возможности графической нотации схем алгоритмов. Представлена система визуального проектирования и генерации программного кода на основе блок-схем и UML. Описаны принципы работы системы, обозначены пути и сферы развития. Приведены примеры и результаты работы разрабатываемой системы. Предлагаемая система может быть использована как в целях обучения, так и в профессиональной программной инженерии.

Ключевые слова: UML, диаграмма деятельности, блок-схема, ГСА, трансляция, генерация программных кодов, проектирование, программная инженерия, Java, Pascal.

Введение. На данный момент существует множество способов графического изображения алгоритма. Среди наиболее известных способов можно назвать следующие: блок-схема, UML-диаграммы (деятельности, состояния, последовательности), драконсхема и диаграмма Насси – Шнейдермана [1].

Анализ существующих графических нотаций бизнес-процессов показывает, что наиболее эффективными с точки зрения отражения концептуальных и реализационных особенностей является модельный аппарат UML. Данный факт в сумме с простотой изучения делает UML наиболее популярным графическим языком описания проектных решений [2].

Особенностью UML-диаграмм есть то, что они поддерживают объектно-ориентированную парадигму. Среди статических моделей объектно-ориентированных программ основной является диаграмма классов. Генерация объектно-ориентированного программного кода по UML-диаграмме классов даёт на выходе так называемый «скелет» программного продукта [3], так как она определяет реализацию конкретных методов, но не позволяет получить полный исполняемый код.

По этой причине большое количество CASE-систем [4], которые используют UML, не позволяют полностью автоматизировать процесс создания программного продукта. Вместе с тем следует отметить, что большой процент затрат в рамках проекта связан именно с кодированием тел методов.

Описание системы. С целью сокращения рутинных трудозатрат на кодирование в процессе создания программного обеспечения, разрабатывается система, позволяющая генерировать исполняемый код по графическому представлению алгоритма [5].

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

Именная типизация означает, что в зависимости от данных, которые хранит в себе переменная, модифицируется её имя. 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.

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.

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).

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

Возможны следующие ошибки структуры ГСА: отсутствие терминальных вершин (начало и конец) или множественность терминальных вершин одного класса, наличие бесконечных циклов, недостижимость вершины из начальной вершины или 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.

Завершающим этапом работы системы является трансляция. 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.

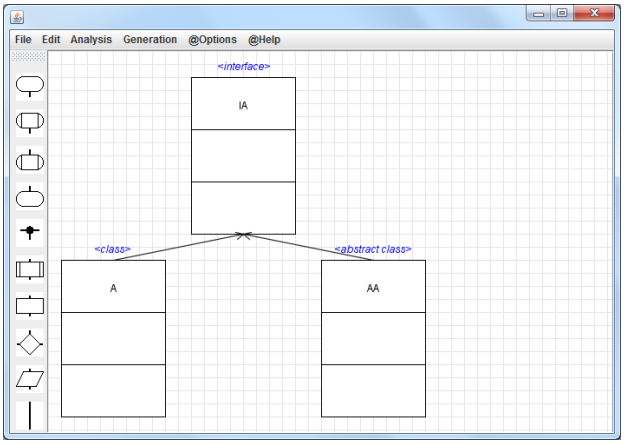
Обсуждение результатов разработки системы. На рис. 1 приведен пример работы разрабатываемой системы. Система позволяет строить иерархию классов и интерфейсов, которая может быть дополнена описанием полей и методов. В интерфейсе системы предусмотрена возможность последующей трансляции сгенерированного промежуточного кода и выполнение его.

На данном этапе разработки системы трансляция сгенерированного кода и выполнение происходят за счёт запуска исполняемого файла операционной системы Windows. Это создаёт зависимость разрабатываемой системы от платформы, на которой она выполняется.

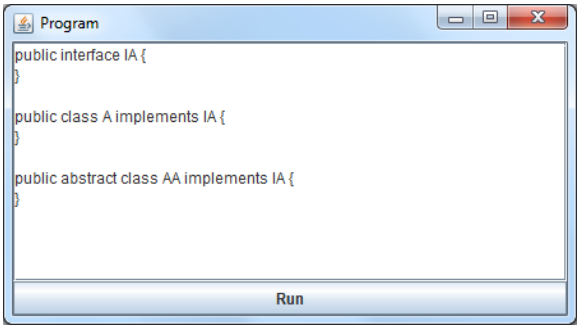
На рис. 2 приведен пример работы системы с генерацией тела метода. В окне запуска, кроме отчёта о трансляции кода, отображается результат выполнения программы (вывод строки на экран).

Сама система разрабатывается на языке Java, что значительно снижает её зависимость от платформы. Прототип системы [5] позволял генерировать код на языке Pascal. В данном варианте системы эта возможность сохранена и расширена для 64-битной операционной системы за счёт использования компилятора Free Pascal (рис. 3).

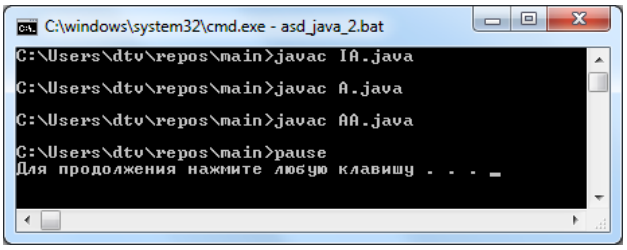
Особую роль в использовании разрабатываемой системы играет возможность инкапсуляции частей алгоритма с помощью сложных блоков (составных операторов). Их применение позволяет сократить общее количество блоков схемы, уменьшить детализацию, абстрагироваться от реализации стандартных структур алгоритма, что приводит к облегчению восприятия блок-схемы в целом. По сути своей сложные блоки есть подпрограммами, которые могут быть созданы путём выделения части существующего алгоритма. На эту часть алгоритма накладывается ограничение на наличие одной точки входа и одного выхода (рис. 4).



а



б

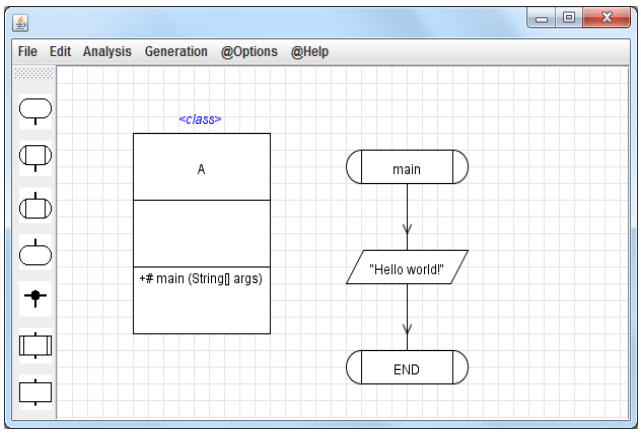


в

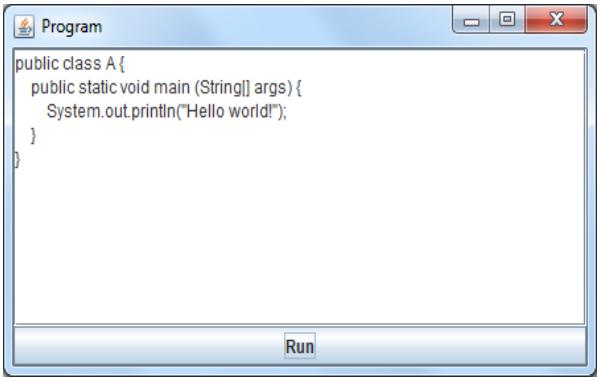
Рис. 1 – Пример работы разрабатываемой системы: а – исходная диаграмма, б - сгенерированный код, в – окно запуска

Такие сложные блоки сохраняются во внешнем хранилище. Хранилище может представлять собой отдельный файл для каждого нового блока или общий файл для нескольких семантически связанных сложных блоков. Такой общий файл можно назвать библиотекой. Сложные блоки могут быть использованы в других алгоритмах путём добавления их из созданных ранее библиотек.

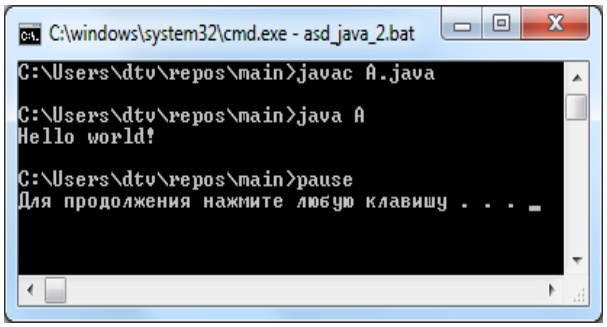
В качестве дальнейшего исследования, полезной функцией для разрабатываемой системы представляется интерактивная связь между графической схемой и сгенерированным кодом. Она позволяет определить фрагмент кода, который соответствует указанным блокам графической схемы. Такая функция позволит существенно ускорить поиск необходимого места в программном коде [6].



а

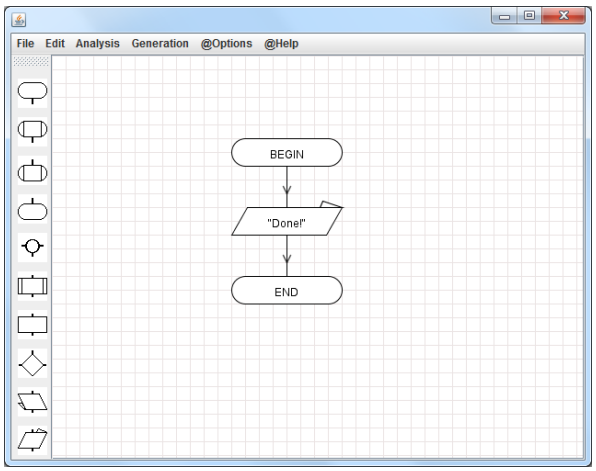


б

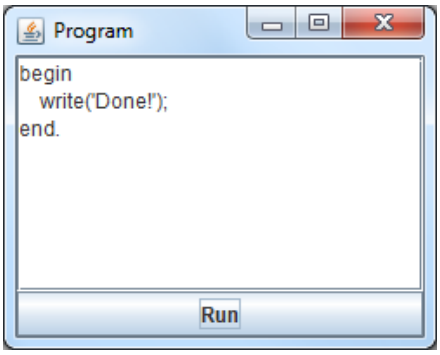


в

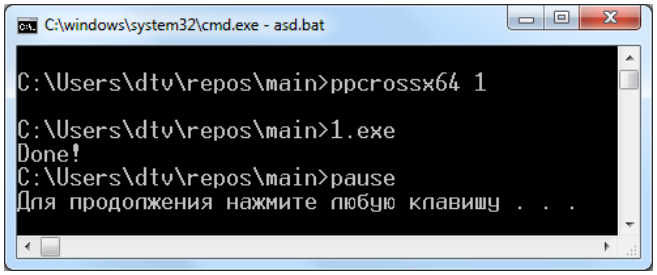
Рис. 2 – Пример генерации тела метода: а – исходная диаграмма, б – сгенерированный код, в – окно запуска



а



б



в

Рис. 3 – Пример генерации кода на языке Pascal: а – исходная блок-схема, б - сгенерированный код, в – окно запуска

Следует отметить, что процесс создания графической схемы естественным образом (рисуя на бумаге) гораздо удобнее по сравнению с манипулированием предопределёнными прототипами блоков и последующим их настраиванием посредством дополнительных окон свойств. Поэтому, система распознавания образов описанных диаграмм с последующей трансляцией их в стандартные примитивы (в терминах базовой системы) является эффективным дополнением с точки зрения пользователя [7-9]. Также полезной функцией системы является возможность импорта диаграмм, созданных в других более распространённых CASE-средствах.

В качестве возможных сфер расширения использования системы можно назвать моделирование бизнес-процессов [4] и предметно-ориентированные языки [10].

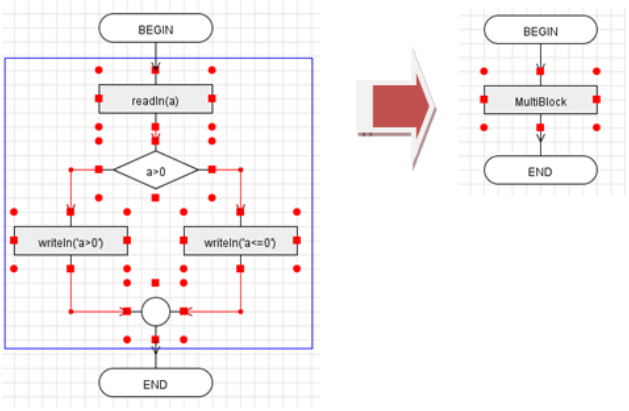


Рис. 4 – Сворачивание набора блоков и связей в один составной блок

Выводы. В статье проанализированы возможности графической нотации схем алгоритмов. Представлена система визуального проектирования и генера-

ции программного кода на основе блок-схем и UML. Описаны принципы работы системы, обозначены пути и сферы развития. Предлагаемая система может быть использована как в целях обучения, так и в профессиональной программной инженерии.

Список литературы: 1. *Дробушевич, Л. Ф.* Способы визуализации алгоритмов и программ [Текст] / *Л. Ф. Дробушевич, В. В. Конах* // Международный конгресс по информатике: информационные системы и технологии. Республика Беларусь, БГУ, Минск, 31 окт. – 3 нояб. 2011г.: в 2 ч. Ч. 1 – Минск: БГУ, 2011. – С. 345–351. – Режим доступа: http://elib.bsu.by/handle/123456789/9836 (22.06.2015) – Загл. с экрана. **2.** *Гайнуллин, Р. Ф.* Разработка методов и средств анализа и контроля диаграмматики бизнес-процессов в проектировании автоматизированных систем: дис. кандидата технических наук : 05.13.12 [Текст] / *Гайнуллин Ринат Фаязович.* – Ульяновск, 2014. – С. 56–57. **3.** *Канжелев, С.* Автоматическая генерация кода программ с явным выделением состояний [Текст] / *С. Канжелев, А. Шалыто* // Paths to Competitive Advantage: Software Engineering Conference. M., 2006. - С. 60–63. **4.** *Аль-Аудат, М. С.* Моделирование бизнес-процессов, CASE − технологии [Текст] / *М. С. Аль-Аудат* // Праці Одеського політехнічного університету: Зб. наук. праць. - Одеса, 2003. - Вип. 1 (19). - С. 306−309. **5.** *Бузовский, О. В.* Система автоматической генерации кодов по графическим схемам алгоритмов [Текст] / *О. В. Бузовский, А. В. Алещенко, А. А. Подрубайло* // Вестник НТУУ "КПИ". Информатика, управление и вычислительная техника. − 2009. − № 51. − С. 204−211. **6.** *Холтыгина, Н. А.* Обзор реализации механизма циклической разработки диаграмм классов и программного кода в современных UML-средствах [Текст] / *Н. А. Холтыгина, Д. В. Кознов* // Системное программирование. Вып. 5: Сб. статей / Под ред. А. Н. Терехова, Д.Ю.Булычева. СПб.: Изд-во СПбГУ, 2010 С. 76−94. **7.** *Александров, А. Е.* Инструментальные средства разработки и сопровождения программного обеспечения на основе генерации кода [Текст] / *А. Е. Александров, В. П. Шильманов* // Бизнес-информатика. - 2012. – №4. – С. 10–17. **8.** *Новиков, Ф. А.* Визуальное конструирование программ [Текст] / *Ф. А. Новиков* // Информационно-управляющие системы. – 2005. – № 6. – С. 9-22. **9.** *С. Прохоренко* PureBuilder. Проект: Среда визуальной разработки без исходного кода, основанная на диалекте языка Oberon [Электронный ресурс] – Режим доступа: https://sites.google.com/site/purebuilder/ (22.06.2015) – Загл. с экрана. **10.** *Дмитриев, С.* Языково-ориентированное программирование: следующая парадигма [Текст] / *С. Дмитриев* // RSDN Magazine – Санкт-Петербург. – 2005. – №5.

Bibliography (transliterated): 1. *Drobushevich, L. F., Konakh, V. V.* (2011). Methods of algorithms and programs imaging. International Congress on Informatics: information systems and technologies. The Republic of Belarus. The Belarusian State University. Minsk, 1, 345–351. **2.** *Gainullin, R. F.* (2014). Development of methods and tools for analyzing and monitoring of business processes diagrammatic in the automated systems design: Dis. candidate of technical sciences: 05.13.12. Ulyanovsk, 56–57. **3.** *Kanzhelev, S., Shalyto, A.* (2006). Automatic program code generation using state. Paths to Competitive Advantage: Software Engineering Conference. Moscow, 60–63. **4.** *Al Audat, M. S.* (2003). Business process modeling, CASE technology. Labor Odessa Polytechnic University: Coll. science papers. Odessa, 1, 306−309. **5.** *Buzovskyy, A. V., Aleshchenko, A. V., Podrubaylo, A. A.* (2009). System of automatic code generation by algorithm graphic schemes. Vestnik NTUU "KPI". Informatika, upravlenie i vychislitel'naja tehnika, 51, 204−211. **6.** *Holtygina, N. A., Koznov, D. V.* (2010). Review of implementation of the mechanism of cyclical development of class diagrams and code in the current UML-tools. Sistemnoe programmirovanie. Saint Petersburg: 5: Sb. Publishing house SPbGU, 76−94. **7.** *Aleksandrov, A. E., Shilmanov, V. P.* (2012). Development tools and support software-based code generation. Biznes-informatika, 4, 10−17. **8.** *Novikov, F. A.* (2005). Visual design of programs. Informatsionno-upravljajuschie sistemy, 6, 9-22. **9.** *Prokhorenko, S.* PureBuilder. Project: Environment of visual development without source code, based on the dialect of the language Oberon. **10.** *Dmitriev, S.* (2005). Linguistic-Oriented Programming: The Next Paradigm. Saint Petersburg: RSDN Magazine, 5.

Поступила (received) 26.05.2015

Відомості про авторів / Сведения об авторах / About the Authors

Бузовский Олег Владимирович – доктор технических наук, профессор, Национальный технический университет Украины «Киевский политехнический институт», профессор кафедры вычислительной техники.
Бузовський Олег Володимирович – доктор технічних наук, професор, Національний технічний університет України «Київський політехнічний інститут», професор кафедри обчислювальної техніки.
Buzovsky Oleg – Doctor of Technical Sciences, Professor, National Technical University Ukraine "Kyiv Polytechnic Institute", Professor of Computer Engineering.
Алещенко Алексей Вадимович – аспирант, Национальный технический университет Украины «Киевский политехнический институт», кафедра вычислительной техники; e-mail: alexey.aleshchenko@gmail.com.
Алещенко Олексій Вадимович – аспірант, Національний технічний університет України «Київський політехнічний інститут», кафедра обчислювальної техніки; e-mail: alexey.aleshchenko@gmail.com.
Aleshchenko Oleksii – graduate, National Technical University Ukraine "Kyiv Polytechnic Institute", Department of Computer Engineering; e-mail: alexey.aleshchenko@gmail.com.

УДК 004.773.5

А. В. ГАБИНЕТ

СПОСОБ МОДЕЛИРОВАНИЯ ПЕРЕДАЧИ ПОТОКОВОГО ВИДЕО В ОДНОРАНГОВЫХ СЕТЯХ

Предложен способ моделирования передачи потокового видео в одноранговых сетях. Предложено использовать топологию сети на основании анализа количества пользователей, которые пользуются услугами доставки потокового видео в зависимости от стран. Описана схема расположения виртуальных машин на одном сервере с помощью виртуализации, а также взаимодействие между ними. Учитываются характеристики сети для эмуляции задержек, потерь пакетов и джиттера.

Ключевые слова: одноранговая сеть, потовое видео, моделирование сети, глобальная сеть, задержка, потери пакетов, джиттер

Введение. Процедура научного исследования требует оценки результатов и методов исследования. Тестирования также должны иметь возможность воспроизводить ся и проверяться другими учеными. Для этого используются три основных способа: аналитическое решение, моделирование, и живой эксперимент [1].

Аналитические решения ссылаются на математическую модель системы, но они хорошо работают только на простых моделях. Симуляторы могут быть использованы для решения неисправности в матема