ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА КОНСТРУИРОВАНИЯ СИНТАКСИЧЕСКИ ПРАВИЛЬНЫХ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ И ПРОГРАММ

Яценко Е.А., Мохница А.С.

Институт программных систем НАН Украины, 03187, г. Киев, просп. Академика Глушкова, 40,
e-mail: yatsenko@mao.kiev.ua
Международный Соломонов университет, 01135, г. Киев, ул. Шолуденко 1б, тел.: 236-11-73,
e-mail: alex_s_m@ukr.net

Рассматривается разрабатываемый интегрированный инструментарий проектирования и синтеза классов алгоритмов и программ в среде Windows. Инструментарий базируется на алгебро-алгоритмических спецификациях и их диалоговой трансформации. Кроме того, он использует метод диалогового конструирования синтаксически правильных программ и включает поддержку современных средств визуализированного проектирования объектно-ориентированных программ (UML, Rational Rose и др.).

The developed integrated means of designing and synthesis of algorithm and program classes in Windows environment are considered. The software is based on algebra-algorithmic specifications and their dialogue transformation. Besides it uses the method of dialogue designing of syntactically correct programs and includes support of contemporary means of visualized designing of object-oriented programs (UML, Rational Rose etc.).

Введение

Важным объектом исследования алгебраической алгоритмики является направление, связанное с дальнейшим развитием соответствующих инструментальных средств. Начало данным средствам было положено разработкой открытых систем (например, МУЛЬТИПРОЦЕССИСТ [1]), базирующихся на схемном проектировании классов алгоритмов и программ.

Данная статья посвящена интегрированному инструментарию проектирования и синтеза классов алгоритмов и программ в среде Windows, который продолжает упомянутые исследования [2, 3, 4, 5].

Разрабатываемый инструментарий базируется на алгебро-алгоритмических спецификациях и их диалоговой трансформации в плане улучшения качественных характеристик (быстродействие, память и т. п.). Кроме того, он включает поддержку современных средств визуализированного проектирования объектно-ориентированных программ (UML, Rational Rose и др.) [6–9].

Необходимо также отметить, что данные инструментальные средства используют метод диалогового конструирования синтаксически правильных программ (ДСП-метод) [10], ориентированный не на поиск и исправление ошибок (как в традиционных синтаксических анализаторах), а на исключение возможности их появления в процессе построения алгоритмов и программ. Инструментарий позволяет строить синтаксически правильные САА-схемы алгоритмов и выполнять синтез программ на современных объектно-ориентированных языках программирования (Java, C++ и др.).

Изложение материала статьи подчинено следующей структуре.

В разделе 1 рассматривается взаимозависимое конструирование аналитических, естественно-лингвистических и граф-схемных проектов.

Раздел 2 посвящен проектированию и синтезу объектно-ориентированных программ.

Вопросам диалоговой трансформации классов алгоритмов и программ символьной обработки, а также редактирования граф-схем алгоритмов, интегрированных с современными системами визуализированного проектирования (UML, Rational Rose) посвящены разделы 3 и 4.

В разделе 5 приведена архитектура инструментальных средств проектирования алгоритмов и программ.

1. Взаимозависимое конструирование аналитических, естественно-лингвистических и граф-схемных проектов

Как уже упоминалось ранее, примером открытых систем схемного проектирования классов алгоритмов и программ является синтезатор МУЛЬТИПРОЦЕССИСТ [1]. Данная система, используя алгоритмические проекты, оформленные на языке САА-схем, генерирует тексты программ на целевых языках программирования (Ассемблер, Си, Паскаль и др.). САА-схемы представляют собой естественно-лингвистические проекты алгоритмов, в основу которых положен аппарат систем алгоритмических алгебр (САА) Глушкова.

Особенность освещаемого инструментария (в отличие от системы МУЛЬТИПРОЦЕССИСТ) состоит в интеграции всех трех представлений алгоритма [10]: аналитического (формула в избранной алгебре), естественно-лингвистического (САА-схемы) и графового (граф-схемы Калужнина) при его конструировании.

Аналитическое представление, как известно, базируется на алгебрах и является компактной записью алгоритма, направленной на его дальнейшее преобразование (минимизация, оптимизация по разным критериям) на базе аппарата соотношений и тождеств, развитых в алгебрах.

Естественно-лингвистическое представление (САА-схема), также базируется на аппарате алгебр, в процессе модификации с помощью метаправил декомпозируется на инвариантную часть (неинтерпретированную схему), которая представляет собою верхний уровень проекта и собственно интерпретации (нижний уровень), зависящие от избранной предметной области. После свертки инвариантная часть может быть снова проинтерпретирована, но уже с помощью других средств нижнего уровня.

Графовое представление, главное преимущество которого – наглядность, также опирается на аппарат алгебр и ориентировано на визуализацию конструируемого алгоритма. Основой для построения этого представления (для аналитического также) может быть выбрана инвариантная часть САА-схемы с дальнейшей интерпретацией в соответствии с избранным нижним уровнем.

Пример 1. Рассмотрим алгоритм асинхронной конвейерной сортировки альтернативными вставками САВ/А, представленный в трех упомянутых выше формах. Пусть М: Н $У_1$ $У_2$ a_1 a_2 ... a_n К – размеченный числовой массив, подлежащий сортировке, где Н и К – маркеры, отмечающие соответственно начало и конец массива М; $У_1$ и $У_2$ – указатели, перемещающиеся по массиву. В процессе сортировки используется также очередь (БС). Аналитическое представление алгоритма имеет следующий вид:

САВ/А ::= СТАРТ * УСТ($У_1$, $У_2$, Н) * ({[d($У_2$, К)] ([l > r| $У_2$] ЗАП.БС(r| $У_2$), Р($У_2$))} * Т(ОБР_ЗАК(1)) * S(ОБР_ЗАК(2)) // {[d($У_2$, К) ∧ ('БС = ∅)] ['БС ≠ ∅] АЛЬТ, Е)} * Т(ОБР_ЗАК(2)) * S(ОБР_ЗАК(1))) * ФИН,

АЛЬТ ::= ЧТЕ.БС(s) * ([l| $У_1$ > s] {[l| $У_1$ < s] L($У_1$)}, {[r| $У_1$ > s] Р($У_1$)}) * ВСТАВ(s, $У_1$).

Здесь СТАРТ – оператор инициализации; УСТ($У_1$, $У_2$, Н) — установка указателей непосредственно за маркером Н; d($У_2$, К) — предикат, истинный, когда указатель $У_2$ достиг маркера К; l| $У_i$ (r| $У_i$) — элемент массива, находящийся слева (справа) от указателя $У_i$; l > r| $У_2$ — предикат, истинный, если указанное отношение выполняется для элементов, стоящих слева и справа от указателя $У_2$; ЗАП.БС(r| $У_2$) – запись элемента r| $У_2$ в основание БС; ЧТЕ.БС(s) – чтение содержимого вершины очереди в переменную s; 'БС = ∅ ('БС ≠ ∅) – условие пустоты (соответственно непустоты) БС; Р, L — сдвиги указателя на один элемент массива вправо и влево соответственно; АЛЬТ – составной оператор, выполняющий поиск посредством перемещения $У_1$ влево или вправо по уже отсортированному фрагменту массива места вставки элемента s из вершины БС, а также осуществляющий вставку s в найденную позицию с помощью оператора ВСТАВ(s, $У_1$); ОБР_ЗАК(i) – условие, истинное, если ветвь i завершила обработку; Т(ОБР_ЗАК(i)) – контрольная точка [11], фиксирующая момент завершения обработки в i-й ветви; S(ОБР_ЗАК(i)) – синхронизатор [11], реализующий ожидание момента завершения вычислений в ветви i, (i = 1, 2); ФИН – заключительный оператор.

Смысл приведенной схемы состоит в асинхронном конвейерном функционировании двух процессов. Первый реализует поиск всех неупорядоченных элементов сортируемого массива с засылкой их в очередь БС, а второй выполняет альтернативную вставку извлекаемых из очереди элементов в уже отсортированную часть массива. Естественно-лингвистическое представление алгоритма САВ/А имеет вид:

```
СХЕМА САВ/А ====
 "Асинхронная конвейерная сортировка альтернативными вставками"
 КОНЕЦ КОММЕНТАРИЯ

"САВ/А"
==== "СТАРТ"
      ЗАТЕМ
      "Установить У(1) в начало (М)"
      ЗАТЕМ
      "Установить У(2) в начало (М)"
      ЗАТЕМ
      (ПОКА НЕ 'Указатель У(2) в конце (М)'
      ЦИКЛ
          ЕСЛИ 'l > r по У(2) в (М)'
          ТО "Записать r по У(2) в (М) в очередь (БС)"
          ИНАЧЕ "Сдвинуть У(2) на (1) по (М) вправо"
          КОНЕЦ ЕСЛИ
      КОНЕЦ ЦИКЛА
      ЗАТЕМ
      КТ 'Ветвь (1) завершила обработку'
      ЗАТЕМ
      ЖДАТЬ 'Ветвь (2) завершила обработку'

      ПАРАЛЛЕЛЬНО

      ПОКА НЕ 'Указатель У(2) в конце (М)'
                  И
                  'Очередь (БС) пуста'
```

```
        ЦИКЛ
            ЕСЛИ 'Очередь (БС) непуста'
            ТО "АЛЬТ"
            ИНАЧЕ "Пустой оператор"
            КОНЕЦ ЕСЛИ
        КОНЕЦ ЦИКЛА
        ЗАТЕМ
        КТ 'Ветвь (2) завершила обработку'
        ЗАТЕМ
        ЖДАТЬ 'Ветвь (1) завершила обработку')
        ЗАТЕМ
        "ФИН"

"АЛЬТ"
==== "Прочитать вершину очереди (БС) в переменную (s)"
        ЗАТЕМ
        ЕСЛИ 'l по У(1) > s в (М)'
        ТО ПОКА НЕ 'l по У(1) < s в (М)'
            ЦИКЛ
            "Сдвинуть У(1) на (1) по (М) влево"
            КОНЕЦ ЦИКЛА
        ИНАЧЕ ПОКА НЕ 'r по У(1) > s в (М)'
                ЦИКЛ
                "Сдвинуть У(1) на (1) по (М) вправо"
                КОНЕЦ ЦИКЛА
        КОНЕЦ ЕСЛИ
        ЗАТЕМ
        "Вставить s слева от У(1) в (М)"
КОНЕЦ СХЕМЫ САВ/А
```

Граф-схема описанной сортировки представлена на рис. 1.

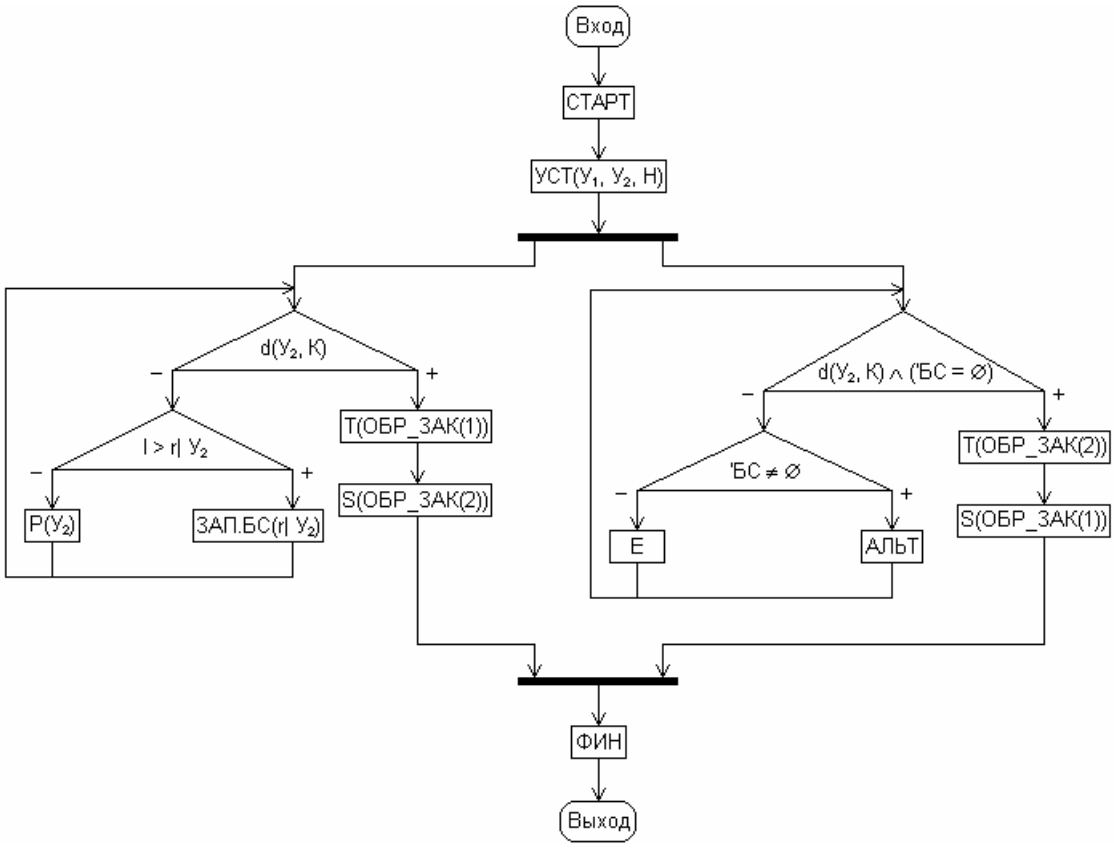

Рис. 1. Граф-схема алгоритма асинхронной конвейерной сортировки альтернативными вставками.

Сделаем ударение на том, что использование в инструментарии всех трех форм представления дает более полную картину о конструируемом алгоритме, чем любое из них по отдельности. Отметим, что изменение любого из этих представлений в инструментарии автоматически приведет к соответствующему преобразованию остальных.

2. Проектирование и синтез объектно-ориентированных программ

Одним из компонентов рассматриваемого инструментария является ДСП-конструктор, ориентированный на автоматизированное проектирование и синтез синтаксически правильных программ. В основу его функционирования положен диалоговый режим с использованием меню подстановок, FIFO (память типа очередь) и дерева конструирования алгоритма. Меню состоит из операторных и логических конструкций, суперпозиция которых позволяет создавать алгоритмы в упомянутых ранее формах (разд. 1). Данные конструкции входят в сигнатуру операций модифицированных САА (САА-М) [11], ориентированных на формализацию последовательных и параллельных вычислений. Выбранные пользователем конструкции, а также операторные и логические переменные, входящие в них, отображаются в дереве с дальнейшей детализацией переменных. В зависимости от типа выбранной переменной система предлагает соответствующий компонент меню или открывает для пользователя архив базисных понятий из базы знаний. Отметим поуровневый стиль конструирования алгоритма, а также возможность переходов на различные уровни (узлы дерева) с продолжением процесса диалогового конструирования, причем подобный переход сопровождается соответствующим изменением состояния FIFO.

С помощью ДСП-конструктора было выполнено диалоговое поуровневое проектирование алгоритма САВ/А (разд. 1). Окно ДСП-конструктора (см. рис. 2) состоит из меню пользователя и трех элементов-окон, двое из которых отображают соответствующие представления алгоритма в процессе его конструирования (САА-схема, формула, дерево алгоритма). Верхнее левое окно предназначено для отображения и выбора элементов подстановки из базы знаний. Дерево алгоритма для каждого составного оператора схемы отображается на отдельной вкладке. На рис. 2 изображены дерево конструирования и САА-схема составного оператора САВ/А.

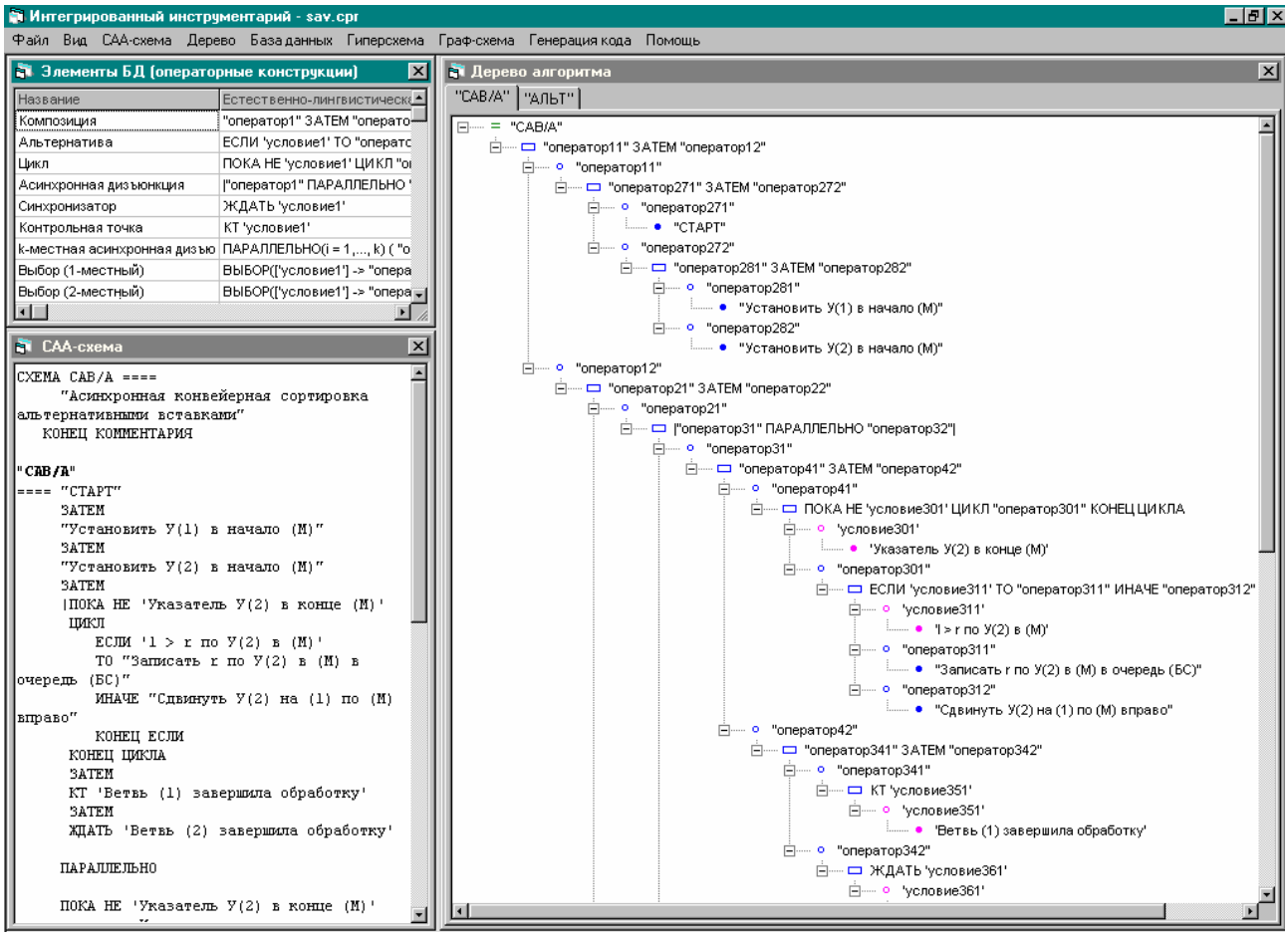

Рис. 2. Результат проектирования в ДСП-конструкторе алгоритма параллельной сортировки альтернативными вставками.

Помимо языковых конструкций и элементарных операторов и предикатов база знаний содержит также стратегии обработки — схемы алгоритмов с переменными. ДСП-конструктор можно использовать как при проектировании новых схем, так и в процессе детализации уже имеющихся стратегий. В дальнейшем

планируется дополнить систему возможностью ввода в нее текста готовых САА-схем алгоритмов. При этом проверка синтаксических ошибок и построение дерева осуществимы с помощью синтаксического анализатора, базирующегося на соответствующих средствах алгоритмики [10].

По дереву алгоритма, реализациям элементарных операторов и условий, а также другим фрагментам программ на целевом объектно-ориентированном языке программирования (Java, C++ и др.), ДСП-конструктор выполняет синтез программы. В процессе синтеза управляющие конструкции схемы отображаются в соответствующие операторы языка программирования, а вместо базисных элементов подставляются их реализации на этом же языке. На вход синтезатора поступает также файл, содержащий каркасное описание основного класса приложения (без реализаций методов), в который выполняется подстановка синтезированного кода. Реализации базисных элементов написаны с использованием программных компонентов для решения задач символьной обработки [5], которые содержат описание данных (массива, указателей, маркеров и др.) и методов доступа к ним. Описать структуру упомянутых классов и выполнить генерацию каркасного программного кода можно с помощью системы Rational Rose (см. разд. 3). Синтез кода для асинхронных алгоритмов связан с использованием потоков (threads) [5, 12].

Предложенный инструментарий был апробирован на последовательных и параллельных алгоритмах сортировки и поиска [5]. Его можно трактовать как средство формализованной технологии повторного использования компонент (ПИК-технологии [6]) для конструирования классов алгоритмов и программ.

3. Диалоговая трансформация аналитических спецификаций алгоритмов символьной обработки

Блок трансформации аналитических спецификаций алгоритмов – одна из важнейших составных частей интегрированного инструментария. Данный блок предназначен для преобразования аналитических спецификаций алгоритмов (формул в разнообразных алгебрах) с использованием соотношений и тождеств, которые содержатся в одном из разделов базы знаний и характеризуют свойства конструкций — операций соответствующей алгебры алгоритмов [10]**.** Упомянутый блок предназначен для выполнения оптимизации формул в соответствии с выбранными критериями (память, быстродействие и т.п.).

Необходимо отметить, что выбор и применение необходимых соотношений и тождеств из базы алгоритмических знаний к формуле происходят в диалоговом режиме, что позволяет пользователю самому выбирать направление и руководить ходом оптимизации.

Отметим, что в основу диалоговой трансформации аналитических спецификаций положены статическая и динамическая декомпозиция цепочек [10], также использующиеся в алгоритмах решения других задач символьной обработки (синтаксического анализа, диалогового конструирования и т.п.).

Аналитические преобразования формул в выбранной алгебре осуществляются с помощью применения равенств, характеризующих свойства операций рассмотренной алгебры.

Как известно, равенства могут применяться к формулам в направления справа, либо в противоположном направлении. В первом случае, в формуле находится подвыражение, статически декомпозируемое по форме, в качестве которой берется левая часть равенства. Полученные в результате декомпозиции значения переменных, входящих в форму, присваиваются соответствующим переменным правой части равенства. Далее найденное подвыражение заменяется подвыражением, сформированным по правой части используемого равенства. Применение равенства в обратном направлении происходит аналогично.

4. Редактирование граф-схем алгоритмов и связь с инструментами визуализации проектирования объектно-ориентированных программ

В данном разделе изложены возможности, предоставляемые одним из основных компонентов инструментария – редактором граф-схем. Также приведены объяснения целесообразности интегрированного использования средств алгебры алгоритмики и объектно-ориентированного проектирования (диаграмм языка UML), причем ударение сделано на визуализационных возможностях граф-схем первой, а также описаны стратегии подобной интеграции.

Одна из главных составных частей инструментария – редактор граф-схем – ориентирована на редактирование внешнего вида граф-схем алгоритмов и программ, которые создаются интегрированным инструментарием в процессе ДСП-конструирования и, возможно, модифицируются в блоке диалоговой трансформации. При этом изменения, внесенные в процессе редактирования, соответствующим образом отображаются на остальных представлениях (аналитическом, естественно-лингвистическом).

Редактор представляет пользователю возможность изменить вид любой компоненты граф-схемного представления, используя: перемещение узлов (преобразователей и распознавателей) граф-схемы с растягиванием дуг, изменение надписей в узлах и их цвета, толщины и стиля линий, размера стрелок. Редактор позволяет добавить на граф-схему ее заголовок и другие надписи. Подчеркнем также возможность повторного использования граф-схем, уже созданных и сохраненных в базе алгоритмических знаний.

Перейдем к взаимосвязи предлагаемого инструментария и средств визуализированного проектирования объектно-ориентированных программ, использующих язык диаграмм UML (например, система Rational Rose).

Основным назначением вышеупомянутого языка является предоставление разработчику средств для моделирования избранной предметной области в терминах классов, их свойств и методов, а также взаимоотношений между этими классами с использованием соответствующих диаграмм [5–8]. Результатом

моделирования в UML является сгенерированный “костяк” программного продукта на выбранном языке программирования. Однако разработкой и кодированием самих методов этих классов приходится заниматься вручную. Помочь разработчику в этом случае, предназначен предлагаемый инструментарий, интегрирующий в себе аналитическое, естественно-лингвистическое и графовое представление алгоритма [2, 3, 4].

Пример 2. Для проектирования и реализации на языке Java алгоритма асинхронной конвейерной сортировки CAB/A (см. пример 1) в Rational Rose были использованы диаграммы классов, деятельности и компонентов [8]. С помощью диаграммы классов описаны типы объектов приложения и отношения, которые существуют между ними. Данная диаграмма содержит класс (Sorting), который служит повторно используемым компонентом при решении различных задач сортировки, а также классы, с помощью которых непосредственно реализуется алгоритм CAB/A (Sav, Thread1 и Thread2), и использующие класс Sorting. Метод main класса Sav выполняет запуск двух ветвей сортировки, код которых описан в классах Thread1 и Thread2. Диаграмма деятельности использована для представления алгоритма сортировки (на ней изображены параллельные ветви обработки и их синхронизация). С помощью диаграммы компонентов представлены компоненты приложения, соответствующие исходным файлам Java, и зависимости между ними. С использованием Rational Rose для компонентов была выполнена генерация каркасного программного кода. Синтез кода, реализующего методы, осуществлен с помощью ДСП-конструктора. Для этого в параметрах генерации алгоритму CAB/A, а также составному оператору АЛЬТ были сопоставлены имена методов, в тело которых ДСП-конструктором был подставлен соответствующий код на целевом языке программирования.

Необходимо отметить, что диаграммы UML, дополненные граф-схемами алгоритмов, существенно обогащают средства визуального проектирования при реализации программных модулей. Благодаря наглядности граф-схем и многоуровневости представления алгоритмов достигается независимость созданного таким образом проекта от кодирования на конкретном языке программирования. Кроме того, на этом этапе при переходе от диаграмм к программному коду может быть использован в качестве средства автоматизации упомянутый интегрированный инструментарий.

Заметим, что дуальной к изложенному подходу является возможность использования средств алгебры алгоритмики (включая граф-схемы и интегрированный инструментарий) на начальных этапах проектирования с дальнейшим использованием диаграмм UML и изложенного выше перехода к объектно-ориентированным программам. Таким образом, речь идет о дуальных стратегиях проектирования объектно-ориентированных программ: “сверху-вниз” и “снизу-вверх”, а также об их комбинированном применении.

Подчеркнем, что необходимость привлечения аппарата граф-схем к средствам UML, нашла отражения в новейших версиях этого языка, что облегчает и обосновывает необходимость создания предлагаемых интегрированных средств [8].

Итак, объединение упоминавшегося инструментария, использующего аппарат граф-схем, и средств визуализированного проектирования объектно-ориентированных программ (UML, Rational Rose) предоставляет пользователю комплект, который позволяет сопровождать разрабатываемый проект на всех стадиях жизненного цикла программной системы.

5. Архитектура инструментальных средств проектирования алгоритмов и программ

Предлагаемый интегрированный инструментарий состоит из компонентов, представленных на рис. 3.

Как упоминалось ранее (разд. 2), с помощью ДСП-конструктора осуществляется диалоговое проектирование и синтез объектно-ориентированных программ с использованием элементов базы знаний.

Блок трансформатора (разд. 3) ориентирован на преобразование схем алгоритмов с использованием соотношений и тождеств, содержащихся в одном из разделов базы знаний. В диалоговом режиме осуществляется выбор тождества или соотношения, необходимого для использования в процессе преобразования. Предназначение данного компонента — оптимизация схемы соответственно избранным критериям.

Следующий блок – редактор граф-схем (разд. 4)**,** ориентирован на редактирование внешнего вида граф-схем, созданных ДСП-конструктором и, возможно, преобразованных блоком трансформатора в процессе проектирования алгоритмов и программ. При этом изменения, внесенные во время редактирования, соответствующим образом отображаются на остальных представлениях алгоритма.

Генератор САА-схем [13] ориентирован на параметрически управляемую генерацию схем алгоритмов и программ посредством спецификаций более высокого уровня, называемых регулярными гиперсхемами (РГС). РГС применяются, в частности, для представления алгоритмов управления выводом в грамматиках структурного проектирования (ГСП) [1]. Проектирование гиперсхем, как и САА-схем, выполняется в диалоговом режиме.

База алгоритмических знаний инструментария вмещает следующие разделы:

- схемы алгоритмов (разработанные алгоритмы из разных предметных областей);
- стратегии обработки (схемы, которые описывают классы алгоритмов и подлежат дальнейшей детализации);
- метаправила свертки, развертки и трансформации (обеспечивают абстрагирование, детализацию и переинтерпретацию схем, а также содержат тождества и соотношения для преобразования схем);
- базисные понятия и их программные реализации (ориентированные на проектирование алгоритмов и синтез программ в данной предметной области на избранном целевом языке);

- графические элементы, используемые для представления граф-схем алгоритмов (различные виды стрелок, блоков и др.).

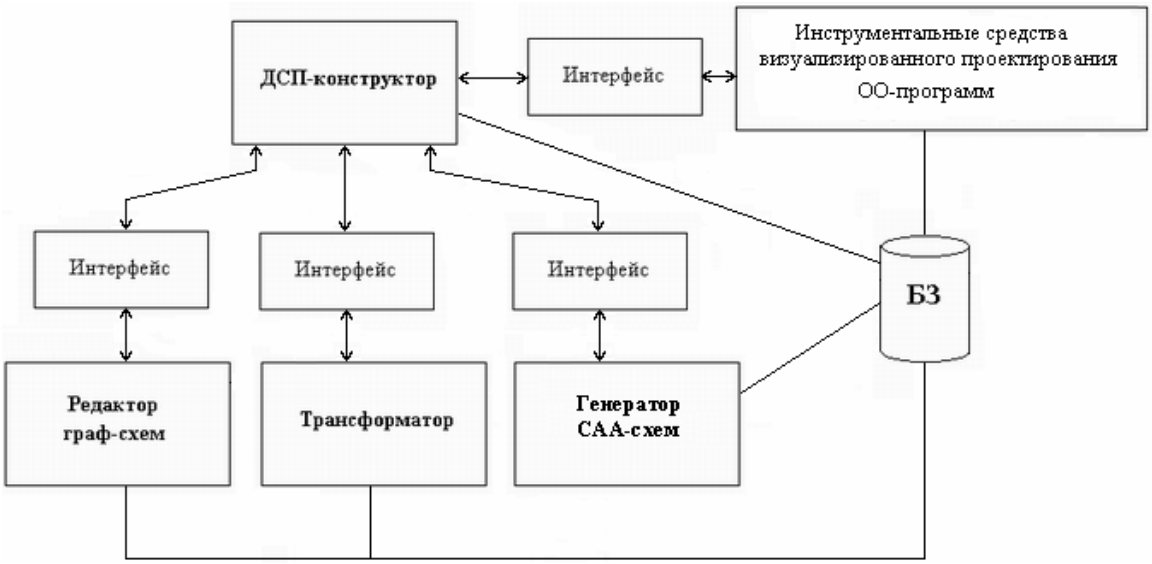

Рис. 3. Архитектура интегрированного инструментария.

Заключение

Подчеркнем, что представленный в статье интегрированный инструментарий диалогового проектирования и синтеза синтаксически правильных объектно-ориентированных программ имеет следующие особенности:

- наличие блока диалоговой трансформации аналитических спецификаций алгоритмов и программ, предназначенного для их оптимизации по избранным критериям (память, быстродействие);
- использование граф-схем как посредника между диаграммами UML и синтезируемым программным кодом;
- возможность конструирования и синтеза параллельных алгоритмов и программ;
- интегрированное использование разработанного инструментария и средства объектно-ориентированного проектирования Rational Rose для автоматизированной разработки объектно-ориентированных программных комплексов любой сложности;
- наличие средств параметрически управляемой генерации параллельных и последовательных схем алгоритмов и программ по гиперсхемам.

Перспективы дальнейших исследований связаны с возможностью разработки и использования в описанном инструментарии разнообразных языковых средств, базирующихся на семействах алгоритмических алгебр (клонах), ассоциированных с современными методами разработки программ (структурным, неструктурным, объектно-ориентированным) [10].

Литература

1. Ющенко Е.Л., Цейтлин Г.Е., Грицай В.П., Терзян Т.К. Многоуровневое структурное проектирование программ: Теоретические основы, инструментарий. – М.: Финансы и статистика, 1989. – 208 с.
2. Цейтлин Г.Е., Амонс А.А., Головин О.В., Зубцов А.Ю. Интегрированный инструментарий проектирования и синтеза классов алгоритмов и программ // Кибернетика и системный анализ. – 2000. – №3. – С. 165–170.
3. Яценко Е.А. Конструирование параллельных объектно-ориентированных программ // Проблемы программирования. – 2002. – №1–2. – С. 188–197.
4. Мохниця О.С. Інструментальні засоби проектування та синтезу синтаксично правильних об’єктно-орієнтованих програм // Вісник Міжнародного Соломонового університету.
5. Цейтлин Г.Е., Яценко Е.А. Элементы алгебраической алгоритмики и объектно-ориентированный синтез параллельных программ // Математические машины и системы. – 2003. – № 2. – С. 64-76.
6. Бабенко Л.П., Лаврищева К.М. Основи програмної інженерії: Навч. посіб. – К:.Т-во "Знання", КОО. – 2001. – 269 с.
7. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++, 2-е изд. / Пер. с англ. – М.: “Издательство Бином”, СПб.: “Невский диалект”, 2000. – 560 с.
8. У. Боггс, М. Боггс. UML и Rational Rose. М.: ЛОРИ. – 2000. – 582 с.
9. UML 2.0 Specification. – http://www.omg.org/technology/uml/ index.htm.
10. Цейтлин Г.Е. Введение в алгоритмику. Киев: Сфера. – 1998. – 310 с.
11. Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л. Методы символьной мультиобработки. – К.: Наукова думка, 1980. – 252 с.
12. Бишоп Д. Эффективная работа: Java 2. – СПб.: Питер, 2002. – 592 с.
13. Яценко Е.А. Алгебры гиперсхем и интегрированный инструментарий синтеза программ в современных объектно-ориентированных средах. – Кибернетика и системный анализ. – 2004. – №1.