На правах рукописи

**БОВКУН**

**Александр Яковлевич**

# ИНСТРУМЕНТАЛЬНАЯ ПОДДЕРЖКА ПОДКЛЮЧАЕМЫХ МОДУЛЕЙ В ЯЗЫКАХ ПРОЦЕДУРНО-ПАРАМЕТРИЧЕСКОГО ПРОГРАММИРОВАНИЯ

05.13.11 – математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

**АВТОРЕФЕРАТ**

диссертации на соискание ученой степени кандидата технических наук

Красноярск – 2010

Работа выполнена в Федеральном государственном автономном образовательном учреждении высшего профессионального образования «Сибирский федеральный университет»

Научный руководитель: доктор технических наук, профессор
**Легалов Александр Иванович**

Официальные оппоненты: доктор технических наук, профессор
**Рубан Анатолий Иванович**

кандидат физико-математических наук, доцент
**Епихин Андрей Михайлович**

Ведущая организация: **Институт систем информатики имени А. П. Ершова СО РАН (г. Новосибирск)**

Защита состоится « **28** » ***января 2011*** **г.** в ***14*** часов на заседании диссертационного совета ДМ 212.099.05 при Сибирском федеральном университете по адресу: г. Красноярск, ул. Киренского, 26, ауд. УЛК 115.

С диссертацией можно ознакомиться в научной библиотеке ФГАОУ ВПО «Сибирский федеральный университет».

Ученый секретарь
диссертационного совета  Непомнящий О.В.

## ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

**Актуальность проблемы.** При разработке программного обеспечения учитываются разнообразные критерии качества. Достижение таких критериев определяется как использованием соответствующей дисциплины программирования, так и применением языковых и инструментальных средств, ориентированных на поддержку избранной парадигмы. Одним из важных критериев качества, актуальных в настоящее время, является эволюционное расширение больших программных систем. Использование эволюционного расширения вносит дополнительную гибкость в создаваемые программные системы и дает возможность применять отлаженный код при разработке новых программ.

Существуют различные подходы, обеспечивающие поддержку эволюционной разработки программного обеспечения. Среди них стоит отметить: объектно-ориентированное программирование (ООП), аспектно-ориентированное программирование (АОП), функциональное программирование и ряд других.

Отдельно стоит выделить процедурно-параметрическое программирование (ППП). ППП, как и ООП, рассчитано на гибкое расширение существующих альтернативных структур, обладающих общим интерфейсом. Но, в отличие от ООП, оно эффективно поддерживает и множественный полиморфизм, что обеспечивает гибкое эволюционное изменение мультиметодов. ППП получило развитие в работах Легалова А.И., Швеца Д.А., Легалова И.А. Были предложены новые методы эволюционного расширения программ за счет использования процедурно-параметрических обобщений и обобщающих параметрических процедур. Помимо этого разработаны новые методы организации структуры программы, позволяющие расширять пространство имен на основе подключаемых модулей.

Вместе с тем, инструментальная поддержка процедурно-параметрической парадигмы до конца не проработана. В частности, не определено влияние методов трансформации модульной структуры программы на формирование процедурно-параметрических отношений во время трансляции, компоновки и загрузки. Полученные в ходе ранее проводимых исследований реализации либо опирались на традиционную модульную структуру, либо предлагаемая структура программы на основе подключаемых модулей в большей степени исследовалась на возможности поддержки новых методов программирования без ее воплощения в реально

выполняемый код. Поэтому способы внутренней организации модулей, а также процедурно-параметрических обобщений, методы преобразования исходных языковых структур в исполняемый код и алгоритмы, используемые в ходе этих преобразований, требуют более детальных исследований.

Несмотря на проработку языковых конструкций и общих методов их создания, отсутствует сравнительный анализ различных методов наращивания базовых программных объектов. В частности, эволюционное расширение на основе подключаемых модулей требует более подробного исследования приемов, обеспечивающих реализацию данного подхода. Отсутствует четкое понимание построения модульных программ в сочетании с процедурно-параметрическим подходом. Недостаточно подробно исследованы приемы и алгоритмы генерации объектного кода и компоновки, несмотря на детальную проработку концепции эволюционного расширения. Не определено, на каких этапах процесса трансляции и каким образом осуществлять преобразование эволюционно написанной процедурно-параметрической программы в обычный процедурный код.

Таким образом, актуальным является анализ методов организации структуры программы с использованием подключаемых модулей и способов ее трансформации в исполняемый код при процедурно-параметрической парадигме программирования.

**Целью работы** является исследование принципов реализации инструментальных средств, обеспечивающих трансляцию эволюционно расширяемых программ, написанных с использованием подключаемых модулей и процедурно-параметрической парадигмы программирования.

Для достижения указанной цели в работе решаются следующие задачи.

1) Исследование связи между способами эволюционного конструирования программных объектов и их преобразованием в машинный код на различных этапах процесса трансляции исходного текста программы.

2) Анализ методов формирования и использования модульной структуры программы в процедурно-параметрических языках программирования.

3) Разработка алгоритмов, обеспечивающих преобразование эволюционно расширяемых процедурно-параметрических программ, использующих подключаемые модули, в исполняемый код.

4) Разработка инструментальных средств, обеспечивающих использование предлагаемых методов и алгоритмов.

**Методы исследования.** В диссертационной работе использовались методы дискретной математики, теории алгоритмов, теории языков и

формальных грамматик, теории разработки трансляторов. Для описания синтаксиса языка программирования применялись диаграммы Вирта и расширенные формы Бэкуса-Наура. Разработка языка программирования опиралась на методы объектно-ориентированного и процедурно-параметрического программирования.

**Научная новизна.**

1) Предложена модель трансформации понятий, описывающая изменение зависимостей между программными объектами в процессе трансляции из исходных текстов в исполняемый код, повышающая эффективность анализа механизмов преобразования программных объектов за счет дополнительной декомпозиции.

2) При помощи модели трансформации понятий проведен анализ и выделены перспективные варианты реализации транслятора с языка процедурно-параметрического программирования.

3) Разработаны методы и алгоритмы, поддерживающие гибкое расширение процедурно-параметрической программы при использовании подключаемых модулей и обеспечивающие построение исполняемого кода в процессе трансляции.

**Практическая ценность.**

1) Уточнен и модифицирован язык процедурно-параметрического программирования, поддерживающий подключаемые модули.

2) Разработан транслятор с процедурно-параметрического языка, обеспечивающий апробацию на практике предлагаемых методов эволюционного расширения программ.

Результаты работы использованы в продолжающихся научных исследованиях, а также в учебном процессе по дисциплинам: «Технология программирования», «Методы программирования», «Формальные языки и трансляторы».

**Положения, выносимые на защиту.**

1) Использование модели трансформации понятий облегчает анализ методов преобразования исходных текстов программы в исполняемый код за счет выделения отдельных фаз преобразования и их независимого анализа.

2) Преобразование подключаемых модулей возможно на разных фазах трансляции и может быть увязано как со статической, так и с динамической компоновкой исполняемого кода.

3) Использование подключаемых модулей в сочетании с процедурно-параметрической парадигмой программирования повышает гибкость процесса разработки программного обеспечения.

**Достоверность и обоснованность результатов** диссертации подтверждаются: исследованием методов эволюционной разработки программ, анализом существующих языковых и инструментальных средств, используемых для разработки программного обеспечения, корректным обоснованием постановок задач, точной формулировкой критериев, исследованием и результатами сравнительного анализа существующих подходов к решению поставленной задачи, опытом применения разработанного инструментария для написания программ в процедурно-параметрическом стиле с поддержкой подключаемых модулей.

**Публикации.** По теме диссертации опубликовано десять печатных работ, из которых одна статья в издании по списку ВАК.

**Личный вклад автора.** Автором предложена модель трансформации понятий, поддерживающая эволюционную разработку процедурно-параметрических программ. Им разработан транслятор языка процедурно-параметрического программирования, реализована поддержка основных методик процедурно-параметрического программирования, проведены исследования возможных способов реализации подхода с использованием подключаемых модулей и расширения модульных структур процедурно-параметрических программ. Основные результаты, изложенные в работе, получены либо непосредственно автором, либо с его участием.

**Апробация работы.** Основные положения диссертации докладывались и обсуждались на: межвузовских научных конференциях студентов, аспирантов и молодых ученых, Красноярск (2007, 2008, 2009, 2010); всероссийских научно-методических конференциях и выставках «Повышение качества высшего профессионального образования», Красноярск (2008, 2009, 2010); 3-й всероссийской конференции «Винеровские чтения», Иркутск, 2009; всероссийской конференции студентов, аспирантов и молодых ученых с международным участием «Молодежь и современные информационные технологии», Томск, 2010.

**Структура диссертации.** Диссертация состоит из введения, четырех разделов, заключения, списка литературы и трех приложений. Работа содержит 124 страницы основного текста, 32 рисунка, 1 таблицу. Список литературы содержит 80 наименований.

## КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

Во **введении** представлены цели и задачи исследования, раскрывается актуальность, новизна полученных результатов, практическая значимость. Приводится перечень вопросов, выносимых на защиту.

В **первом разделе** рассматриваются методы трансформации понятий. Проводится анализ эволюционного расширения программных объектов на различных фазах трансляции программы. Для выявления специфики процесса преобразования программы и анализа изменения ее структуры в ходе трансляции из исходного текста в исполняемый код предлагается модель трансформации понятий.

Модель определяет процесс, состоящий из фаз, определяющих поэтапное преобразование исходного текста программы в исполняемый код, и зависимости между программными объектами (понятиями), существующими в промежуточных состояниях программы между этими фазами. Модель задается вектором

$$M = <P_{i,1}, P_{i,2}, \ldots , P_{i,j}, \ldots, P_{i,n}>,$$

где M – модель трансформации понятий, $P_{i,1}$, $P_{i,2}$, ... , $P_{i,j}$, ..., $P_{i,n}$ – фазы процесса трансформации программы, определяемые для i-й системы программирования, содержащей некоторый язык, который поддерживает соответствующие парадигму и технологию программирования, и инструментальные средства, обеспечивающие процесс преобразования программы. Парадигма определяет, каким образом формируется исходный текст программы, а технология задает специфику фаз преобразования и их последовательность.

Фаза трансляции $P_{i,j}$ задает отображение входного представления модели зависимостей программы $S_{i,j-1}$ в выходное представление $S_{i,j}$:

$$P_{i,j}: S_{i,j-1} \rightarrow S_{i,j}.$$

Исходя из сказанного следует, что i-я система программирования, обеспечивающая процесс преобразования исходных текстов программы в исполняемый код, имеет свой набор фаз, зачастую отличающийся по составу от других систем программирования.

На основе анализа модификации понятий в процессе трансформации исходных текстов программы показано, что разработка программного обеспечения и процесс его трансляции в исполняемый код связаны с постоянной трансформацией внутренней структуры программы. В ходе преобразования осуществляется построение языковых конструкций, осуществляющих дополнительную инструментальную поддержку такого критерия качества разработки программного обеспечения, как эволюционное расширение. Преобразование программных объектов может быть

обусловлено как адаптацией к особенностям исполнителя программ, так и тем, что исходный текст программы не может быть напрямую выполнен из-за наличия непрямых связей между понятиями, раскрытие которых ведет или к трансформации этих понятий, или к генерации дополнительного кода, обеспечивающего программное формирование этих связей.

Каждый из языков программирования имеет свой набор фаз и формируемых промежуточных программных объектов, что определяется спецификой поддерживаемых им парадигм. Набор фаз также зависит от подходов, применяемых при трансляции и компоновке. В качестве примера рассматриваются типовые фазы преобразования программы для языка программирования С (рисунок 1).

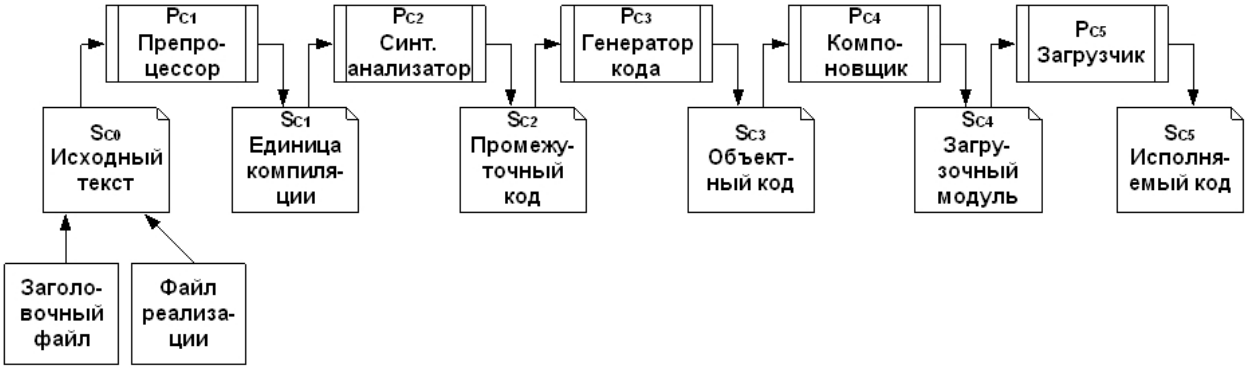



Рисунок 1 – Фазы преобразования программы языка программирования С

Первая фаза использует исходные тексты программы, распределенные по заголовочным файлам и файлам реализации, обеспечивая их препроцессорную обработку. В результате изменяется модульная структура исходного текста, так как описания, расположенные в отдельных единицах, переносятся в общую с функциями единицу компиляции.

На второй фазе происходит раздельная компиляция полученных единиц. Для каждой единицы компиляции формируется единица промежуточного кода, который в ходе следующей фазы преобразуется в машинный код с сохранением числа преобразованных единиц и без изменения сформированных зависимостей между программными объектами.

На фазе компоновки происходит слияние отдельных объектных единиц в единый код и формирование окончательных физических зависимостей путем замены перекрестных внешних ссылок между модулями на физические адреса.

Связывание объектов программы осуществляется с помощью процесса компоновки. В ходе этого процесса выстраиваются дополнительные связи и разрешаются внутренние противоречия, формируются перекрестные ссылки.

На этапе загрузки исполняемый модуль загружается в процессор. Выполнение модуля начинается с процесса инициализации, в ходе которого происходит динамическое формирование внутренних связей.

Приводится описание модели зависимостей понятий как составной части модели трансформации. Модель зависимостей определяет наличие между программными объектами отношений, фиксирующих их расположение относительно друг друга, а также наличие ассоциативной или косвенной связи между ними.

Для описания зависимостей был предложен графический язык, который облегчает интерпретацию и понимание общей структуры программных объектов. Элементы языка отражают:

– имена создаваемых понятий;

– операции по формированию программных объектов;

– связи, определяющие зависимости между объектами;

Выделяются две категории отношений между программными объектами:

1) иерархические отношения реализуются за счет вложения одних объектов в другие;

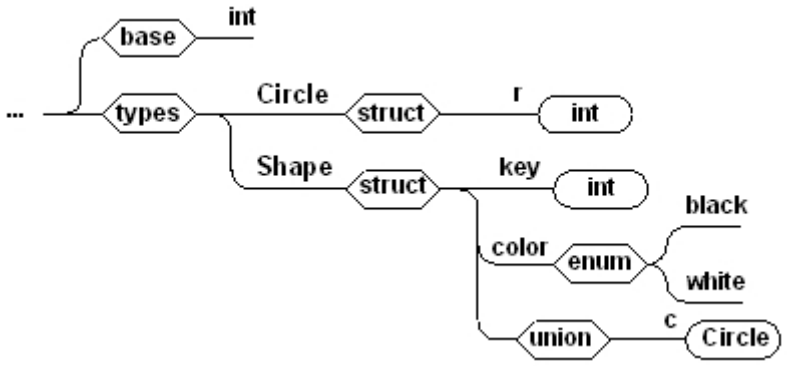
2) отношения на основе ассоциаций описывают косвенную связь на понятия, определенные в других частях модели.

Для оценки возможностей применения модели зависимостей рассматривается расширение обобщения с помощью дополнительной специализации. В качестве примера ниже представлен фрагмент исходного текста процедурной программы, оперирующей геометрическими фигурами. В первоначальном варианте обобщение всех фигур содержит одну специализацию:

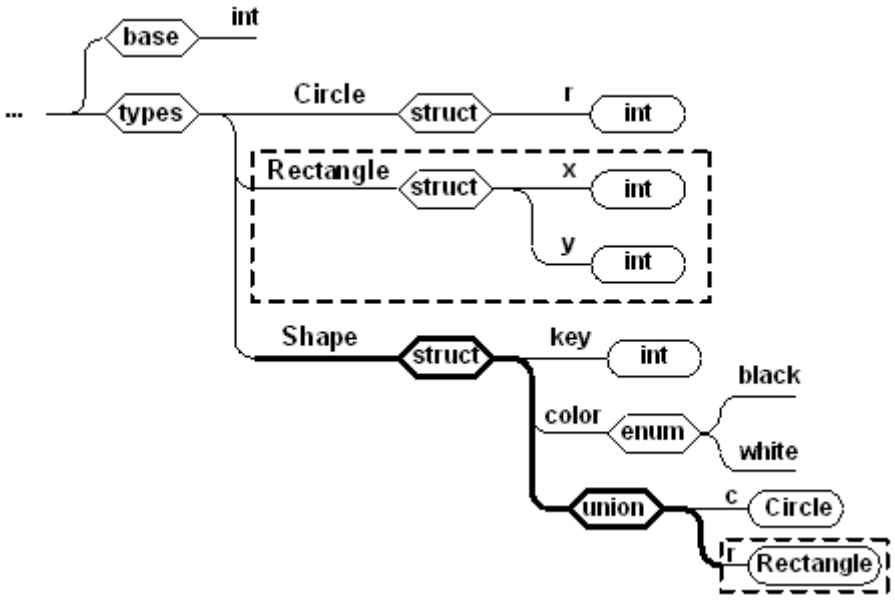
```
// Основа специализации «Круг»
struct Circle {
     int r;
};

// Обобщение «Фигура»
struct Shape {
     int key;
     enum color = {BLACK, WHITE};
     union {
               Circle c;
     };
};
```

Модель зависимостей понятий, соответствующая приведенному коду, отображает взаимосвязь описанных программных объектов (рисунок 2, *а*). Обобщение и специализация описываются независимо друг от друга, а их связь представляется в виде ссылки.




*а*




*б*

Рисунок 2 – Модель зависимостей понятий:
*а* – начальная; *б* – модифицированная

Расширим существующее обобщение за счет добавления прямоугольника. Это приведет к изменению обобщения **`Shape`**, а также модуля, в котором оно находится. Помимо этого добавляемое описание прямоугольника также изменяет этот модуль:

```
// Основа специализации «Круг»
struct Circle {
     int r;
```

```
};

// Добавляемая основа специализации «Прямоугольник»
struct Rectangle {
     int x, y;
};

// Обобщение «Фигура»
struct Shape {
     int key;
     enum color = {BLACK, WHITE};
     union {
               Circle c;
               Rectangle r;
     };
};
```

Изменения в исходном тексте программы отображаются на модели зависимостей, показывая как модификацию иерархических отношений между программными объектами (изменение обобщения показано толстой черной линией, добавление новой специализации – пунктиром), так и перестройку косвенных связей (создание в обобщении ссылки на новую специализацию отмечено пунктиром) (рисунок 2, б).

Показано, что подобные модели можно создавать для каждой из фаз преобразования. В результате трансформаций происходит построение новых программных объектов, что обеспечивает устранение семантического разрыва между исходными текстами программ и формируемым объектным кодом, который после проведенных преобразований может содержать программные объекты, отличающиеся от первоначально написанных. Изучение зависимостей на каждой фазе позволяет понять, является ли система эволюционно расширяемой, а также то, каким образом и на каких этапах осуществляется трансформация зависимостей в ходе дальнейшей трансляции программы.

Во **втором разделе** рассматриваются методы организации и трансформации модульных структур программ. Проводится разделение языков программирования на две группы с точки зрения поддержки модульности.

1) Первая группа использует иерархическое взаимодействие модулей, при котором вновь формируемый модуль может использовать понятия из ранее разработанных модулей. В этом случае все разработанные программные объекты имеют известную структуру, а их имена доступны для

организации ассоциативного или косвенного связывания. Основным достоинством данного подхода является простота формирования программы из законченных логических единиц, для объединения которых не требуются дополнительные сборщики проектов. Это позволяет легко реализовать как статическую, так и динамическую сборку программы, а также дает возможность повторно использовать модули при создании новых программ.

2) Вторая группа языков программирования ориентирована на использование в качестве модулей единиц компиляции, обеспечивающих свободное размещение различных программных объектов, включая и ассоциативные (косвенные) ссылки на еще не созданные понятия. В этом случае раздельная компиляция осуществляется без окончательного формирования структуры для некоторых из косвенно представленных программных объектов. Процесс их формирования переносится на более поздние этапы статической или динамической компоновки.

Эволюционно расширяемые программы могут включать модули, напрямую не связанные с другими модулями прямыми иерархическими отношениями. Поэтому их добавление в программу описывается не в уже существующих модулях (что вело бы к модификации написанного кода), а в отдельно выделенном файле. Для эволюционно расширяемых программ используется в основном второй подход.

С целью поиска возможных вариантов организации модульной структуры программы, повышающих эффективность использования эволюционного расширения, проводится анализ трансформации понятий в модульных языках программирования. Сравниваются модели зависимостей понятий языков программирования Оберон/Оберон-2 и O2M.

Языки семейства Оберонов обладают иерархической модульной структурой, которая не предоставляет прямую поддержку мультиметодов, обеспечивающих использование множественного полиморфизма, из-за необходимости вставлять в ранее написанные модули информацию о вновь создаваемых модулях.

Основным отличием модели зависимости O2M (описание языка и его особенностей представлено в работах Швеца Д.А.) является возможность расширения процедурно-параметрических обобщений и обобщающих параметрических процедур в новых модулях. Вместе с тем, использование обычных модулей в качестве расширителей уже существующих затрудняет понимание программы. Образуется множество мелких по размеру и функциям модулей, которые не несут свою целевую нагрузку. Помимо этого ссылки на обобщенные понятия из других модулей затрудняют восприятие программы из-за наличия в разных модулях собственных пространств имен.

Для устранения недостатков языка в диссертационной работе Легалова И.А. был предложен механизм подключаемых модулей. Этот механизм предполагает использование уже существующего пространства имен при расширении понятий в ранее сформированных модулях. Основной проблемой при его реализации является построение алгоритмов, которые обеспечивали бы трансформацию модульной структуры программы и программных объектов.

Для анализа различных вариантов трансформации процедурно-параметрической программы были рассмотрены модели трансформации ранее существующих языков программирования и зависимости, выстраиваемые между программными объектами в ходе трансляции. В результате был уточнен и модифицирован процедурно-параметрический язык программирования Alien. Язык программирования Alien является расширением предыдущей разработки языка процедурно-параметрического программирования O2M. Во многом сохраняя языковые конструкции O2M, Alien расширяет возможности эволюционного расширения с помощью поддержки обобщенных записей и подключаемых модулей (рисунок 3).

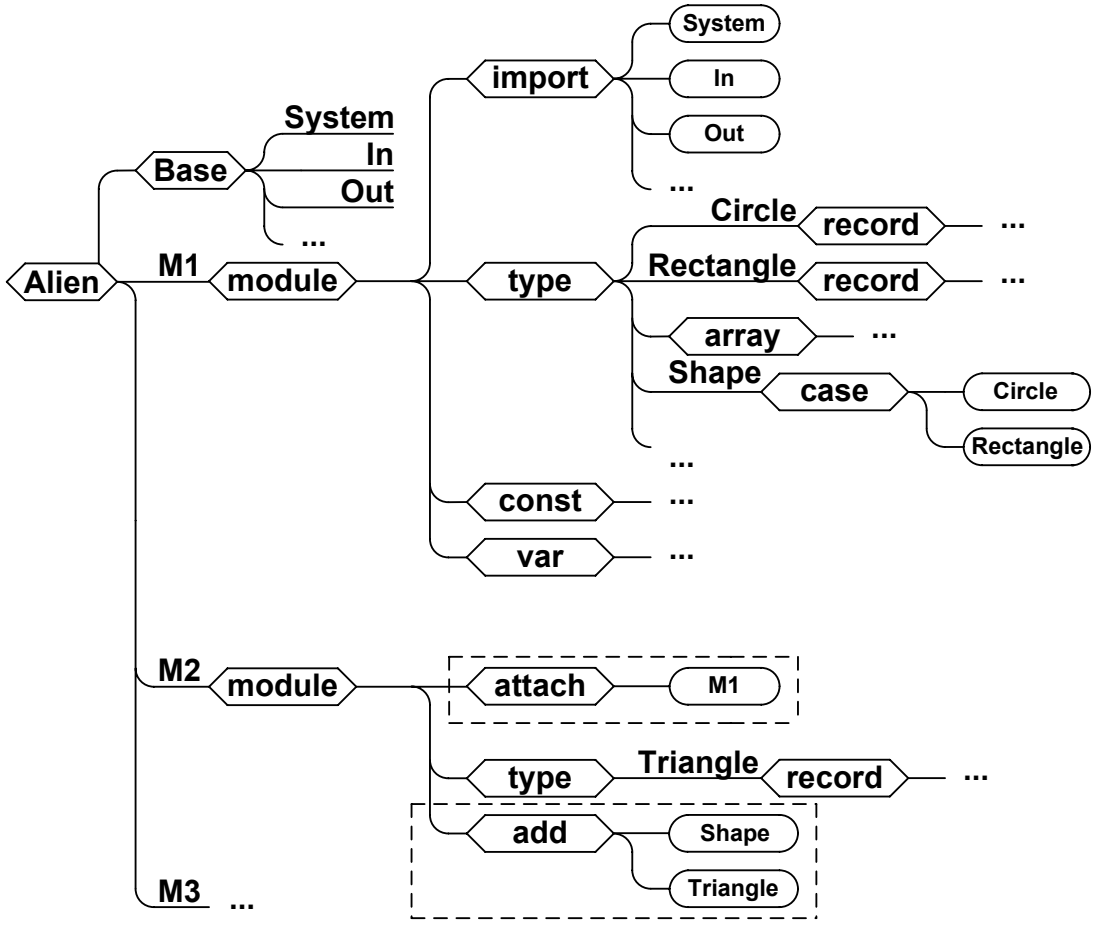



Рисунок 3 – Обобщенная модель зависимостей модульной структуры языка программирования Alien

Рассмотрены различные варианты трансформации программных объектов во время трансляции предложенного языка программирования, среди которых можно отметить следующие:

– использование общего промежуточного представления;
– применение раздельной обработки модулей;
– подход на основе динамически загружаемых модулей.

При использовании варианта на основе общего промежуточного представления (рисунок 4) компиляция модулей осуществляется последовательно друг за другом. В результате компиляции создается промежуточное представление модуля и интерфейс модуля на языке Alien. Сформированный интерфейс используется при импорте модуля в ходе последующей компиляции.

После завершения компиляции и связывания всех модулей программы происходит компоновка промежуточных представлений модулей в единый монолит. Полученное промежуточное представление используется для последующей генерации объектного кода программы, после чего осуществляется сборка и формирование загрузочного модуля.

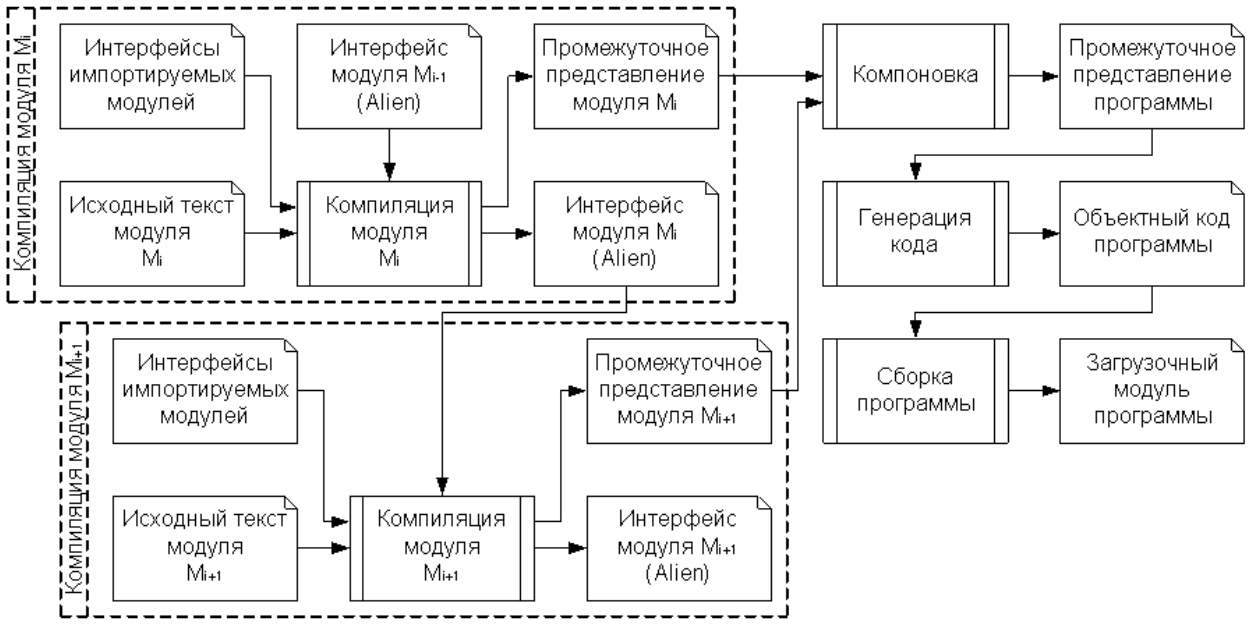



Рисунок 4 – Вариант реализации методов трансформации исходного текста программы на основе общего промежуточного представления

К достоинствам такого подхода следует отнести то, что на данном уровне абстракции проще всего проводить преобразования, так как представления программы на каждом из этапов практически тождественны друг другу. Далее, результатом является монолитная архитектура модулей,

которая в дальнейшем может непосредственно трансформироваться в традиционные языки. Дальнейшие преобразования осуществляются без трансформации, т. е. последующая генерация кода производится для традиционной машины.

К недостаткам подхода стоит отнести то, что библиотечные модули необходимо формировать и хранить в промежуточном виде из-за невозможности их повторного использования на различных этапах.

Второй вариант основан на раздельной обработке модулей (рисунок 5). В этом случае для каждого из модулей программы также происходит построение промежуточного представления. Далее на его основе происходит генерация объектного кода модуля. Помимо промежуточного представления компилятором также создается интерфейс модуля. После обработки всех модулей происходит компоновка объектных кодов в единый код на традиционном языке.

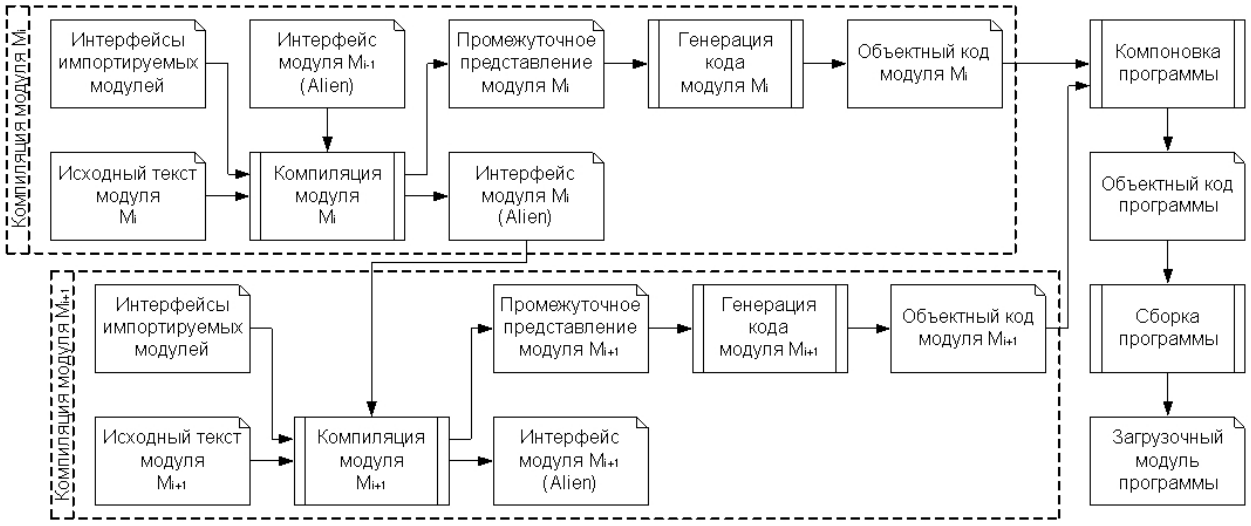



Рисунок 5 – Вариант реализации методов трансформации исходного текста программы на основе раздельной обработки

Отличие от предыдущего подхода заключается в сдвиге момента создания общего промежуточного представления программы, а также его вид (традиционный язык программирования или какой-либо другой).

Преимуществом такого подхода является простота реализации. Генерация промежуточного представления и объектного кода осуществляется для каждого модуля в отдельности и не требует дополнительной компоновки.

Недостатком подхода следует отметить то, что слияние объектных кодов модулей в единый код осуществляется на конкретном объектном языке и может требовать дополнительного анализа и проверки.

При использовании варианта на основе динамически загружаемых модулей (рисунок 6) создание монолитного представления программы сдвигается по времени еще дальше. Для каждого из модулей последовательно выполняются фазы компиляции, построения промежуточного представления, генерации объектного кода и создания загрузочного модуля. Связь модулей осуществляется как с помощью интерфейсных единиц, так и с помощью загрузочных модулей, которые подключаются динамически.

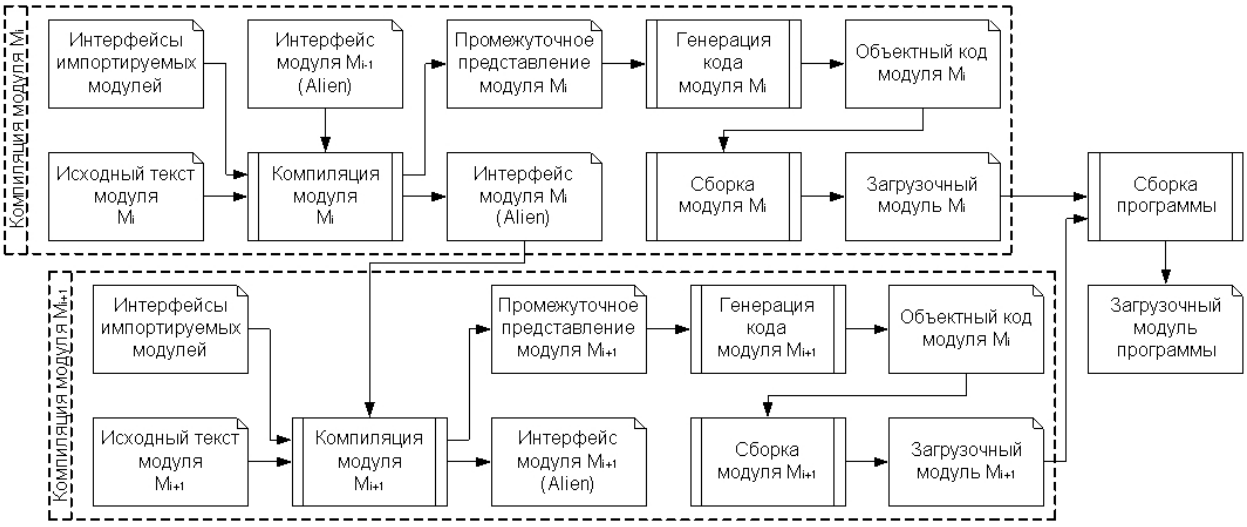



Рисунок 6 – Вариант реализации методов трансформации исходного текста программы на основе динамически загружаемых модулей

Достоинством такого подхода является то, что к этапу формирования загружаемого модуля завершена компоновка модулей, определены взаимосвязи между программными объектами, сформирована окончательная модель программы. Динамическая загрузка позволяет подгружать импортируемые модули только в случае необходимости. Если же модуль не используется никакими другими компонентами, он может быть выгружен из памяти.

К недостаткам подхода следует отнести необходимость строгой проверки интерфейсов загружаемого модуля на совместимость со всеми клиентскими модулями. Если модули-клиенты не совместимы с новой версией модуля, то он не может быть загружен в память до тех пор, пока все клиенты не будут адаптированы к его новой версии (как минимум, перекомпилированы).

Проанализированные подходы в организации этапов трансляции процедурно-параметрической программы обладают своими достоинствами и

недостатками. Для устранения недостатков отдельных вариантов можно использовать комбинированные подходы. Такие подходы могут представлять различные комбинации как вышеописанных, так и других методов организации.

В результате получаемая структура программных объектов производится с помощью различных методов и может образовываться на самых разных этапах трансляции: во время сборки проекта, при компиляции физических единиц, в ходе компоновки или динамической загрузки.

В **третьем разделе** исследуется влияние подключаемых модулей на реализацию процедурно-параметрического подхода в языке программирования Alien. На основе анализа модульной структуры, организации параметрических обобщений, обобщенных записей, а также обобщающих параметрических процедур и обработчиков специализаций предлагаются алгоритмы, позволяющие повысить гибкость использования подключаемых модулей в процедурно-параметрическом подходе.

Разработанные алгоритмы в ходе трансляции преобразуют исходную, эволюционно расширяемую, модель зависимостей понятий процедурно-параметрической программы (рисунок 3) в модель зависимостей, используемую в исполняемом коде (рисунок 7).

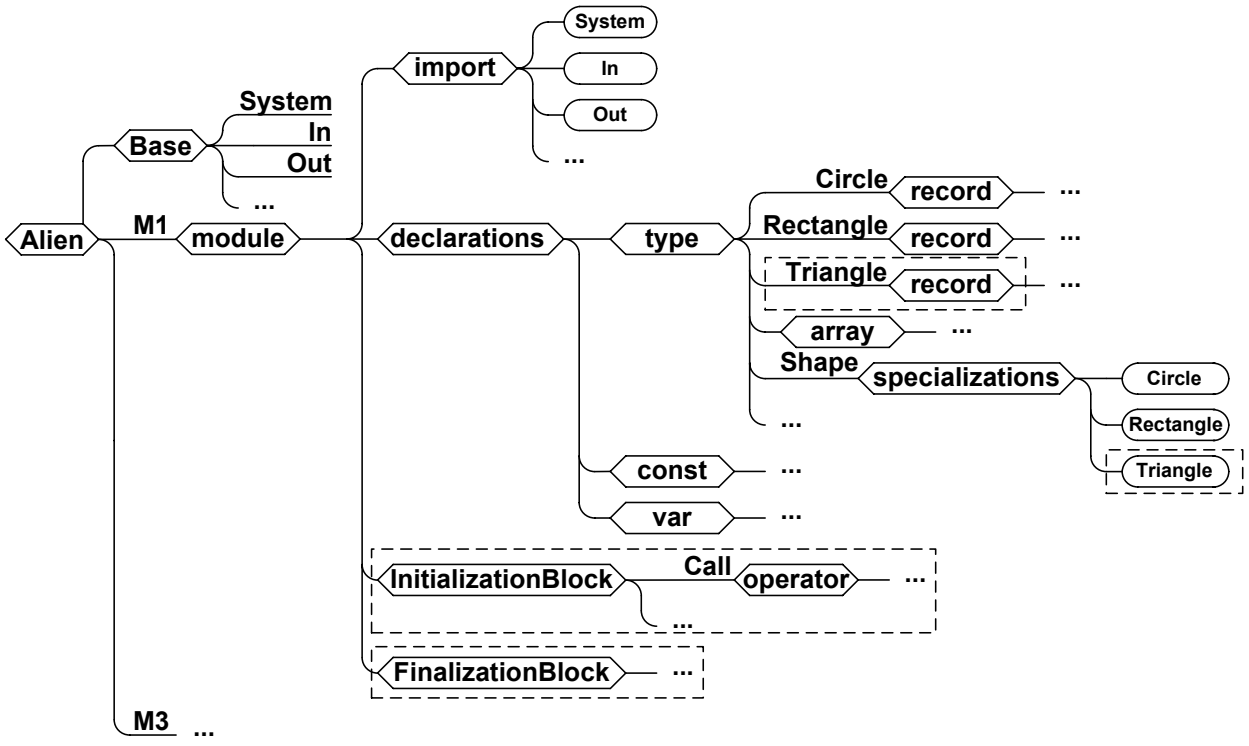



Рисунок 7 – Обобщенная результирующая модель зависимостей модульной структуры языка программирования Alien

В ходе трансформации модулей осуществляется объединение объектов пространств имен базового и подключаемого модулей. В частности, описание специализации обобщения, введенное в подключаемом модуле, перемещается в базовый модуль. Операция расширения обобщения заменяется на прямое включение специализации в базовом модуле. Помимо этого явно описываются блоки инициализации и финализации, так как построение модели осуществляется после компиляции программы. Список импортируемых единиц остается неизменным. Трансформация остальных понятий программы осуществляется согласно введенным алгоритмам.

Получаемая в итоге модель программных объектов может использоваться для дальнейшего анализа или генерации объектного кода программы для ее выполнения с помощью традиционных компиляторов. Генерация объектного кода в традиционные языки программирования позволит провести дополнительное сравнение исходных текстов и итогового представления программных объектов.

**Четвертый раздел** посвящен особенностям разработки процедурно-параметрического языка, использующего подключаемые модули. Для практической апробации изменений в языке программирования Alien разработаны необходимые инструментальные средства. В их состав вошли:

– транслятор с языка программирования Alien;

– компоновщик параметрических отношений.

Разработка транслятора обеспечила практическую поддержку экспериментов, связанных с исследованием механизма подключаемых модулей и методов трансформации программы из исходного текста в исполняемый код.

Рассмотрены объектная модель транслятора и его промежуточное представление. Проанализированы особенности реализации в трансляторе многомерных массивов данных. Для создания и хранения многомерных массивов с неизвестной заранее размерностью предлагается использовать одномерный массив. При хранении данных в одном большом одномерном массиве минимизируются накладные расходы на память, на время создания массива, на время доступа к отдельному элементу.

В **приложении А** приведено описание процедурно-параметрического языка программирования.

В **приложении Б** приведено соответствие конструкций процедурно-параметрического языка и элементов его промежуточного представления, получаемого после этапа трансляции.

В **приложении В** приведен пример программы, демонстрирующий использование подключаемых модулей и процедурно-параметрических обобщений для гибкого эволюционного расширения уже написанного кода.

## ОСНОВНЫЕ РЕЗУЛЬТАТЫ И ВЫВОДЫ

1 Предложена модель трансформации понятий, описывающая изменение зависимостей между программными объектами в процессе трансляции из исходных текстов в исполняемый код, повышающая эффективность анализа механизмов преобразования программных объектов за счет дополнительной декомпозиции.

2 При помощи модели трансформации понятий проведен анализ и выделены перспективные варианты реализации транслятора с языка процедурно-параметрического программирования.

3 Разработаны методы и алгоритмы, поддерживающие гибкое расширение процедурно-параметрической программы при использовании подключаемых модулей и обеспечивающие построение исполняемого кода в процессе трансляции.

4 Уточнен и модифицирован язык процедурно-параметрического программирования, поддерживающий подключаемые модули.

5 Разработан транслятор с процедурно-параметрического языка, обеспечивающий апробацию на практике предлагаемых методов эволюционного расширения программ.

**Результаты работы отражены в следующих публикациях:**

**1 Бовкун, А. Я. Расширение модульной структуры программы за счет подключаемых модулей / А. Я. Бовкун, А. И. Легалов, И. А. Легалов // Доклады академии наук высшей школы Российской Федерации. – Новосибирск : НГТУ, 2010. – С. 114–125.**

2 Бовкун, А. Я. Методы динамической классификации при работе с информацией / А. Я. Бовкун // Повышение качества высшего профессионального образования : материалы Всерос. науч.-метод. конф. с междунар. участием : в 2 ч. Ч. 1. – Красноярск : СФУ, 2007. – С. 270–271.

3 Бовкун, А. Я. О моделировании процесса конструирования программных объектов (г. Красноярск) / А. Я. Бовкун // Материалы 3-й Всерос. конф. «Винеровские чтения» [Эл. ресурс]. – Иркутск, 2009.

4 Бовкун, А. Я. Инструментальная поддержка динамических классификаторов / А. Я. Бовкун // Молодежь и наука: начало XXI века : материалы III Всерос. науч.-техн. конф. студентов, аспирантов и молодых ученых : в 4 ч. Ч. 1. – Красноярск : СФУ, 2007. – С. 75–78.

5 Бовкун, А. Я. Создание программных объектов в инструментальных средах / А. Я. Бовкун // Молодежь и наука: начало XXI века : сб. материалов Всерос. науч.-метод. конф. студентов, аспирантов и молодых ученых : в 5 ч.

Ч. 2 / сост. О. А. Половинкина; МИОЦ ФГОУ ВПО СФУ. – Красноярск, 2008. – С. 14–16.

6 Бовкун, А. Я. Методы динамической классификации при работе с информацией / А. Я. Бовкун // Повышение качества высшего профессионального образования : материалы Всерос. науч.-метод. конф. : в 2 ч. Ч. 1 / науч. ред. В. И. Колмаков; отв. за вып. С. А. Подлесный. – Красноярск : СФУ, 2008. – С. 216–218.

7 Бовкун, А. Я. Обучение современным методам разработки программного обеспечения на основе обобщенной модели конструирования понятий / А. Я. Бовкун, А. И. Легалов // Повышение качества высшего профессионального образования : материалы Всерос. науч.-метод. конф. : в 3 ч. Ч. 1 / отв. ред. С. А. Подлесный. – Красноярск : ИПК СФУ, 2009. – С. 66–69.

8 Бовкун, А. Я. Модель конструирования понятий как основа модульной структуры программы / А. Я. Бовкун // Проблемы информатизации региона. ПИР-2009 : материалы одиннадцатой Всерос. науч.-практ. конф., Красноярск, 2–3 ноября 2009 г. / под ред. д.т.н. Г. А. Доррера; отв. за выпуск к.т.н. Л. Д. Якимова. – Красноярск : РИЦ СибГТУ, 2009. – С. 105–107.

9 Бовкун, А. Я. О моделировании процесса конструирования программных объектов / А. Я. Бовкун // Молодежь и наука: начало XXI века : сб. материалов Всерос. науч.-метод. конф. студентов, аспирантов и молодых ученых : в 5 ч. Ч. 2 / сост. О. А. Половинкина; МИОЦ ФГОУ ВПО СФУ. – Красноярск, 2009.

10 Бовкун, А. Я. Исследование характеристик модульной структуры программы / А. Я. Бовкун, И. А. Легалов // Молодежь и современные информационные технологии : сб. тр. VIII Всерос. науч.-практ. конф. студентов, аспирантов и молодых ученых «Молодежь и современные информационные технологии», Томск, 3–5 марта 2010 г., ч. 2. – Томск : Изд-во СПБ Графикс. – 132–133 с.

Подписано в печать 23.12.2010
Формат 60x84/16. Усл. печ. л. 1,1
Тираж 100 экз. Заказ № 2895

Отпечатано в полиграфическом центре Библиотечно-издательского комплекса
Сибирского федерального университета
660041, г. Красноярск, пр. Свободный, 82а