Технологический инструментарий для построения систем анализа и преобразования структуры программ

К. С. Стефанов*

В данной статье описывается предлагаемая архитектура технологического инструментария для построения систем анализа структуры программ. Инструментарий предназначен для построения специализированных систем для решения конкретных задач, возникающих при исследовании последовательных программ и их преобразовании с целью исполнения на компьютерах с параллельной архитектурой.

Описываются принципы разбиения инструментария на логические уровни: внутреннее представление, зависимые и независимые от языка модули, уровень модулей целевых функций (экспертов) и уровень интерфейсов. Определяются возможные взаимодействия между модулями инструментария, расположенными на разных уровнях, с целью локализации зависимости от языка анализируемой программы и ОС выполнения инструментального комплекса. Рассмотрено разделение модулей на модули анализа и преобразований. Рассматриваются отдельные уровни с примерами расположенных на них модулей.

1. Введение

При создании программного обеспечения для вычислительных систем с параллельной архитектурой не только сохраняются все

*Научно-исследовательский вычислительный центр МГУ

трудности традиционного программирования, но и добавляется множество новых, связанных с организацией параллельного выполнения программы.

Ручное распараллеливание последовательных программ — занятие крайне трудоемкое. Некоторые методы автоматического распараллеливания реализуются в компиляторах для параллельных вычислительных систем. Отличительная особенность всех этих методов — они должны работать быстро, поэтому не могут быть сложными. В работе [1] проанализированы методы выявления параллелизма, заложенные в компиляторы для параллельных систем начала 90-х годов прошлого века. Оказалось, что сфера применения каждого из методов очень узка. В целом же при анализе зависимостей различных операторов компиляторам удается лишь в 15% случаев без трудностей определить характеристики этих зависимостей. Несмотря на то, что за прошедшие 10 лет появилось много новых способов распараллеливания вычислений, в общем ситуация изменилась не сильно.

Для облегчения создания параллельного программного обеспечения создаются системы автономного анализа и преобразования программ. На данный момент существует некоторое количество проектов, нацеленных на облегчение создания параллельного ПО. Это семейства SUIF, KAP, пакеты FORGE, Polaris, CAPO, система ParaWise и другие. Часть этих систем является автоматическими препроцессорами, другие имеют возможности по интерактивному просмотру результатов анализа и получению подсказок по преобразованию. Некоторые рассчитаны только на машины с общей памятью, и не имеют функциональности для работы с распределением данных. В пакете FORGE реализовано распараллеливание под компьютеры с распределенной памятью, но с довольно жесткими ограничениями на используемое распределение данных и возможную конфигурацию параллельных циклов.

При этом исходный код большинства этих систем недоступен, и использование их функций для решения других задач невозможно.

В настоящее время в НИВЦ МГУ ведутся работы по созданию технологического инструментария, который позволил бы конструи-

ровать системы анализа и преобразования программ с необходимой функциональностью на основе готовых блоков.

2. Принципы построения технологического инструментария

В основу инструментария закладываются три принципа: локализация зависимостей от конкретного языка анализируемой программы, модульность инструментария и гибкость архитектуры.

Под гибкостью мы понимаем возможность последующего построения на основе инструментария специализированных систем и конверторов, предназначенных для решения максимально широкого спектра задач, которые возникают в процессе адаптации программ под параллельные вычислительные системы: определение потенциального параллелизма программ, выделение параллельных участков кода, преобразование программ под конкретную систему параллельного программирования и т.п. Разбиение на модули осуществляется так, чтобы иметь возможность из уже написанных модулей технологического инструментария конструировать варианты новых систем для разных задач. Например, это может быть препроцессор, цель которого в полностью автоматическом режиме выявить весь имеющийся в программе параллелизм и записать его в виде комментариев специального вида. Или наоборот, это может быть система с графическим интерфейсом, показывающая пользователю структуру исследуемой программы и позволяющая ему самому провести необходимые преобразования.

Инструментарий сразу проектируется с расчетом на будущее добавление анализируемых языков процедурного типа. Именно поэтому отделяются модули, которые потребуется доработать при переходе к анализу программ на других языках.

Модульность не только облегчает построение специализированных систем с нужной функциональностью, но и позволяет использовать в качестве модулей блоки, взятые из других систем, а также внешние программы, реализовав интерфейс с ними в виде модуля-оболочки инструментария.

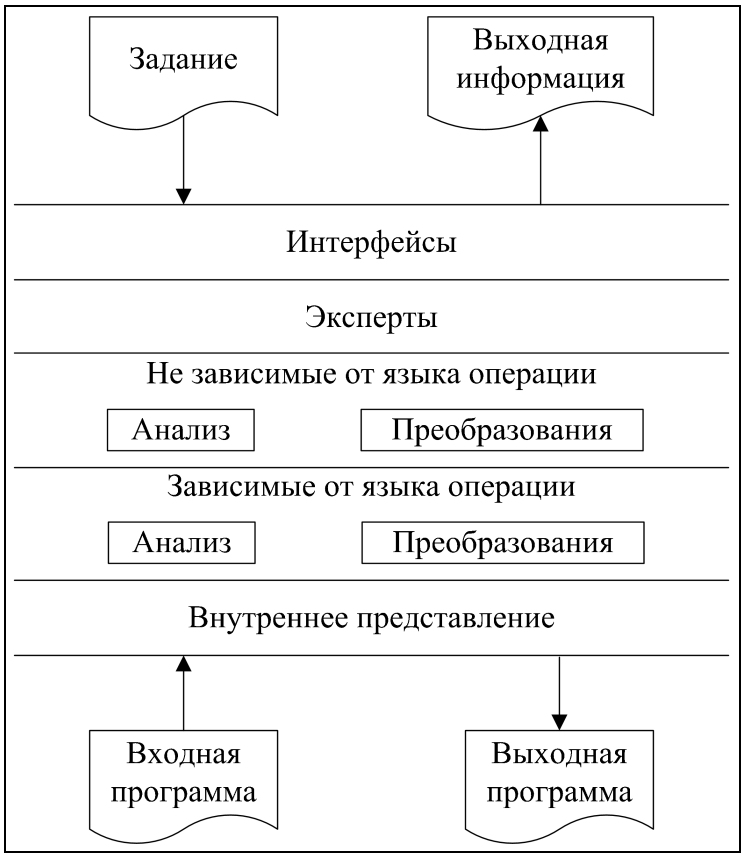

Рис. 1. Логические уровни блоков инструментального комплекса

3. Общая структура инструментария

Все блоки системы логически разделяются на уровни (рис. 1): *внутреннее представление, зависимые от языка* (входной программы) блоки, *независимые от языка* блоки, *эксперты* и *интерфейсы*.

Деление на уровни определяет внутреннюю структуру комплекса и ограничивает возможные взаимодействия блоков. С блоками

внутреннего представления могут взаимодействовать только блоки, зависимые от языка. Интерфейс, предоставляемый зависимыми от языка блоками верхним уровням, скрывает особенности конкретных языков и детали внутреннего представления программ.

Уровни зависимых и независимых от языка операций дополнительно делятся на блоки определения свойств (анализ) и блоки преобразований.

Эксперт — это блок, имеющий целевую функцию, нужную пользователю системы. Например, эксперт может выявлять все потенциально параллельные циклы программы и отражать эту информацию в комментариях, или осуществлять преобразование программы под конкретную систему параллельного программирования.

Интерфейс предоставляет доступ к функциям системы внешним пользователям, в роли которых могут быть и другие программы. Все взаимодействие комплекса или созданной на его основе системы с внешним миром — с пользователем либо с другими программами — происходит через интерфейсы.

4. Уровень внутреннего представления

Уровень внутреннего представления составляют блоки, отвечающие непосредственно за работу с текстом программы (ее разбор или генерацию нового текста) и за действия со структурами данных, составляющих представление разобранной программы.

Блоки этого уровня в самой большой степени зависят от языка анализируемой программы. Именно на этом уровне осуществляется разбор текста входной программы. Структуры данных для разных входных языков могут сильно отличаться, однако интерфейс, предоставляемый блокам верхнего уровня, должен быть выражен в терминах описания языков. Поскольку мы ориентируемся прежде всего на процедурные языки, эти термины должны быть похожи для разных языков, естественно, в той степени, в какой похожи языки программирования процедурного типа.

Интерфейс блоков уровня внутреннего представления позволяет производить различные действия с объектами программы такими как операторы, выражения, переменные, программные едини-

цы, файлы программы (если входной язык предусматривает деление программы на файлы).

Например, в Си есть один тип программной единицы — функция, в Фортране-77 типов программных единиц несколько: процедура, функция, главная программа и т.д., а в Фортране-90 добавляется еще модуль. В Си есть глобальные переменные, а в Фортране-77 связь между программными единицами по общим данным осуществляется при помощи COMMON-блоков. Интерфейс блоков внутреннего представления должен отражать эти и другие особенности каждого анализируемого языка.

Возможности по генерации программ должны давать возможность генерировать выходную программу из любых конструкций. При этом корректность генерируемой программы на данном уровне может проверяться лишь частично, оставляя такую проверку зависимым от языка модулям преобразований.

5. Блоки анализа и преобразований

Уровни зависимых и независимых от языка блоков содержат основные подсистемы комплекса — анализ и преобразование программ, и предоставляют функциональность, необходимую для построения экспертов.

Блоки анализа служат для определения свойств входной программы, которые затем могут быть использованы другими блоками для выполнения своих функций. Это могут быть свойства как отдельных объектов программы (является ли данный оператор оператором перехода, имеет ли данная переменная составной тип), так и свойства фрагментов программы (принадлежит ли данный фрагмент программы к линейному классу [2].

Блоки анализа программ на выходе дают объекты-свойства, содержащие информацию о входной программе. Это могут быть элементарные объекты логического типа, описывающие наличие или отсутствие в анализируемой программе какого-либо признака, например, возможность распараллеливания конкретного цикла. Также это могут быть сложные объекты, например, граф вызовов программы.

Программный интерфейс для работы с такими объектами должен быть определен либо в терминах анализируемых языков, если объект-свойство существует на уровне между внутренним представлением и зависимыми от языка блоками, либо в терминах, общих для анализируемых языков, если такой объект существует на уровнях выше уровня зависимых от языка блоков.

Блоки преобразований служат для выполнения базовых преобразований программы, которые могут быть использованы в качестве составных частей другими блоками преобразований или модулями-экспертами. Отнесение того или иного преобразования к уровню зависимых или независимых от языка блоков либо же включение его как составной части в какой-нибудь эксперт определяется тем, может ли данное преобразование использовано другим экспертом. Многие преобразования широко используются при различных стратегиях распараллеливания под разные вычислительные системы. В то же время существуют и преобразования, использование которых вряд ли может выходить за пределы достаточно узкой области. Например, при преобразовании программы под систему параллельного программирования OpenMP имеет смысл определить преобразование «добавление директивы OpenMP», опирающейся на добавление комментария или директивы #pragma. Использование комментариев для задания директив распараллеливания в программах на Фортране применяется не только в OpenMP, а также может применяться и для записи в тексте программы каких-либо найденных ее свойств. Поэтому добавление комментария целесообразно сделать модулем преобразования.

Деление на зависимые и независимые от языка определяется возможностью определить свойство или преобразование в терминах, общих для всех анализируемых языков. Подробнее про это деление будет сказано ниже.

6. Зависимые от языка блоки

Зависимые от языка блоки — это блоки определения свойств или проведения преобразований, которым нужно знать специфику языка, на котором написана анализируемая или преобразуемая

программа. Эти блоки работают через интерфейс, предоставляемый блоками уровня внутреннего представления.

Каждый такой блок должен быть повторен для каждого анализируемого входного языка. При этом интерфейс самих этих блоков, предоставляемый блокам верхних уровней, должен выражаться в терминах, общих для всех анализируемых языков программирования, а выбор блока для конкретного языка должен обеспечиваться самим инструментарием.

Зависимые от языка блоки анализа служат для определения тех свойств программы, для которых нужно знать специфику языка, на котором она написана. Например, для построения графа управления программы нужно уметь определять, в каком порядке передается управление между операторами. Определение того, какой оператор может быть выполнен после данного, является блоком, реализация которого зависит от языка анализируемой программы. В Фортране есть управляющие операторы GOTO, IF, оператор цикла DO. В языке Си набор управляющих операторов другой: goto, if, switch, break, continue и операторы цикла for, while, do ... while. Наборы управляющих операторов различаются от языка к языку.

Зависимые от языка блоки преобразований, по аналогии с зависимыми от языка блоками анализа, определяют преобразования, специфические для данного языка. Например для добавления новой переменной нужно создать оператор, ее описывающий, и вставить в блок описаний (если входной язык требует описаний переменных). При этом информация о возможных типах переменных, а также о расположении блока описаний, структуре самих описаний и т.п. зависит от языка программирования, на котором записана преобразуемая программа.

Некоторые преобразования программы можно делать более эффективно, если не требовать сохранения корректности программы на промежуточных этапах (разумеется, полное преобразование должно сохранять корректность). Например изменение размерности массива, используемое иногда для приведения программы к линейному виду [3], может привести к тому, что операции доступа к элементам этого массива станут некорректными. Можно разработать

последовательность преобразований, включающую введение нового массива и последовательную замену всех обращений к одному массиву на обращения к другому, но проще временно отключить проверку полной корректности получаемой программы после каждого шага преобразования (если такая проверка вообще проводится). Поэтому необходимо иметь возможность отключения некоторых проверок на корректность. Естественно, что ответственность за полную корректность получаемой программы лежит на программисте, реализующем преобразования в таком режиме. При этом синтаксическая корректность получаемой программы гарантируется средствами генерации программы уровня работы с внутренним представлением.

7. Независимые от языка блоки

Независимые от языка блоки анализа определяют те, обычно более «высокоуровневые», свойства, которые можно определить общим для всех языков процедурного типа образом, опираясь на информацию, поставляемую зависимыми от языка блоками. Информация, необходимая для работы данных блоков, выражается в терминах, общих для всех анализируемых языков.

Например построение графа управления процедуры может быть осуществлено опираясь на информацию, какой оператор может быть выполнен после данного. Получив информацию о переходах между операторами, граф управления в терминах отдельных операторов может быть построен без учета особенностей конкретных языков.

Независимые от языка блоки преобразования предоставляют возможности по преобразованию программы, необходимые блокам экспертов. По аналогии с независимыми от языка блоками анализа, обычно на данном уровне проводятся более «высокоуровневые» преобразования чем те, которые проводятся на уровне зависимых от языка преобразований.

Практически все составные преобразования должны находиться на данном уровне. При их проведении нужно опираться на «элементарные» преобразования, проводимые зависимыми от языка блоками преобразований.

В блоках экспертов всегда используется информация, поставля-

емая блоками независимого от языка уровня. Если эксперту необходима информация от блока зависимого от языка уровня или уровня внутреннего представления, то надо создать дополнительный блок-прослойку, передающий такую информацию наверх. Такая архитектура облегчит решение проблем, возникающих при добавлении новых анализируемых языков.

В некоторых случаях полезно иметь набор блоков, имеющих одинаковый интерфейс и выполняющих одну функцию разными методами. Например, одним из методов, используемых для приведения программы к линейному виду, является априорная подстановка [4]. При этом значение одной переменной заменяется на некоторое выражение. Такая замена может производиться, например, модулями, определяющими используемые переменные или индексные выражения на лету, прозрачно для вызывающих модулей. Таким образом реализация использования подстановки сведется к изменениям в небольшом количестве модулей анализа и написании модуля уровня экспертов, определяющего правила такой подстановки.

8. Уровень модулей целевых преобразований (экспертов)

Блоки, реализующие целевую функциональность системы или конвертора, расположены на уровне экспертов. Эксперт — модуль системы, реализующий конкретную нужную пользователю функцию в автоматическом или полуавтоматическом режиме, например, распараллеливающий и преобразующий программу под конкретную систему параллельного программирования.

В рамках рассматриваемого технологического инструментария вся целевая функциональность системы оформляется модулями уровня экспертов. Эти модули могут быть как весьма простыми, например, вызывающими какой-либо блок анализа для всех операторов программы и таким образом собирающим статистику. Или же эксперт может быть сложным, реализующим множество функций по приведению программы к виду, пригодному для использования в определенной системе параллельного программирования и осу-

ществляющий ее оптимизацию.

Различные модули уровня экспертов могут взаимодействовать с модулями своего уровня или модулями независимого от языка уровня.

В некоторых случаях эксперту может потребоваться дополнительная информация об анализируемой программе помимо содержащейся в ее тексте. Это может быть, например, информация о входных и выходных данных какой-либо подпрограммы или информация о параллельных свойствах какого-либо фрагмента программы, не поддающегося анализу. В этом случае эксперт запрашивает эту информацию у вызывающего его модуля (это может быть как другой эксперт, так и модуль уровня интерфейсов), который и сообщает ему требуемое. При вызове эксперта, которому могут понадобиться дополнительные данные, вызывающий модуль предоставляет информацию, каким образом получать требуемые сведения (или о том, что надо пользоваться значениями по умолчанию).

Таким образом может быть реализована, например, используемая в системе V-Ray схема специальных комментариев [2]. Такие комментарии размещаются в тексте программы и сообщают анализирующей системе о некоторых свойствах этой программы, например, о границах изменений каких-либо переменных. В этом случае эксперт, использующий такую информацию в своей работе, запрашивает ее у вызывающего модуля. Этот модуль может быть другим экспертом, который просмотрит имеющиеся в программе комментарии и сообщит требуемую информацию (либо о ее отсутствии).

9. Уровень интерфейсов

Блоки интерфейсов предназначены для реализации различных способов взаимодействия построенной системы с внешним миром, например, графического интерфейса пользователя или интерфейса командной строки, который может использоваться как человеком, так и другими программами.

У каждой системы, созданной при помощи инструментария, должен быть один основной блок интерфейса. Этот блок должен определить и сообщить другим блокам системы:

- На каком языке написана анализируемая программа и какие файлы ей принадлежат.
- Какие эксперты будут вызываться после загрузки анализируемой программы.
- Какие еще модули интерфейсов будут использоваться.
- Куда будет записан текст сгенерированной программы и другие выходные файлы (если они есть).
- Другую информацию, необходимую вызываемым экспертам.

Как видно, основной блок интерфейса является центром общения системы с внешним миром. При этом информация для каждого из пунктов может быть получена практически независимо. Язык анализируемой программы и имена файлов могут быть получены у пользователя в окне-диалоге, могут быть параметрами командной строки или содержаться в пакетном файле описания задачи.

Другие используемые интерфейсные модули и блоки-эксперты скорее всего будут иметь различные программные интерфейсы, поэтому их использование определяется программным кодом данного модуля интерфейса. Для каких-то конкретных систем может использоваться ровно один эксперт, собирающий какую-нибудь статистику по свойствам анализируемого программного комплекса. В этом случае модуль интерфейса будет достаточно простым без использования других интерфейсов. Или наоборот, у сложной системы с графическим интерфейсом может быть предоставлен выбор по вызываемым экспертам, а для настройки каких-то экспертов могут привлекаться другие интерфейсные модули. Например, в рамках одной системы могут одновременно присутствовать возможности распараллеливания программы и под OpenMP, и под MPI, оформленные в виде разных экспертов. В этом случае основной блок интерфейса должен предоставить пользователю выбор, какой эксперт запускать, а настройки разных экспертов будут производиться соответствующими интерфейсными модулями.

Определение местоположения выходных файлов (сгенерированной или преобразованной программы и других файлов с полученной информацией) может определяться любым из способов, которым определяется расположение входных файлов. Могут быть заданы какие-то умолчания или, например, схема получения имени выходного файла из имени входного (замена суффикса и т.п.).

Как уже отмечалось выше, некоторым экспертам при работе может потребоваться дополнительная информация. Она может позволять провести более точный анализ или иногда указать эксперту про какие-то свойства программы, известные пользователю, но которые не удается получить при анализе текста программы. Если при этом эксперт вызывается напрямую из интерфейса (или через цепочку других экспертов, которым неоткуда взять эту информацию), эксперт делает запрос к вызвавшему его модулю о наличии требуемой информации, передавая сведения о каком объекте программы и что именно он хочет узнать. Интерфейс инициирует диалог с пользователем с запросом соответствующих сведений, если такая возможность предусмотрена, или сообщает, что данных у него нет. Также информация может быть получена не от пользователя а, например, из файла, описывающего задание для системы. Но в этом случае, особенно если таких сведений может быть много, целесообразно вынести эту информацию в отдельный файл и создать еще один интерфейсный модуль, читающий данный файл и сообщающий сведения запрашивающему модулю.

При построении пользовательского интерфейса к объектамсвойствам программы добавляются *объекты-представления* этих свойств. Представление используется блоком интерфейса при выдаче описания свойства пользователю, причем одному свойству может соответствовать множество представлений. Например, свойство параллельности цикла может быть представлено текстовым комментарием к заголовку этого цикла, директивой OpenMP или же разметкой вершины, соответствующей этому циклу на графе управления. В последнем случае мы имеем дело с представлением внутри другого представления.

Блоки, вырабатывающие представление по объекту, являются

составными частями интерфейсов. Поскольку одно и то же представление может требоваться в разных интерфейсах, такие блоки представлений выделяются отдельными блоками для возможности повторного использования.

Некоторым представлениям может требоваться дополнительная информация. Например, одним из представлений графа зависимостей программы может служить не его проекция на ось операторов программы, а полное изображение в пространстве итераций. При этом конкретный размер изображения (границы изменений параметров циклов) зависит от значений внешних переменных. В этом случае либо должны быть предусмотрены какие-то разумные значения по умолчанию, либо надо спрашивать конкретные значения у пользователя.

В блоках уровня интерфейсов должна быть локализована зависимость от среды, в которой исполняется система. Именно блоки этого уровня знают о желательном расположении файлов, о вызовах, требуемых для открытия этих файлов. Другим блокам при этом передаются объекты языка программирования, на котором они написаны, представляющие открытый файл (например объект типа istream или ostream Си++).

В этих же блоках локализуется зависимость от способа общения с пользователем, будь то интерфейс командной строки или графическая среда. Существует множество вариантов библиотек для построения графических интерфейсов, и каждая из них имеет некоторый ограниченный набор платформ, на который она может быть задействована. Поэтому локализация зависимости от конкретной библиотеки облегчает перенос полученной системы под другую ОС.

При построении систем анализа крайне важно тестирование корректности их работы. Если набор реализованных функций велик, то ручное тестирование становится весьма трудным, если не невозможным. При этом в процессе разработки необходимо регулярно проводить максимально полное тестирование, чтобы быть уверенным, что вносимые изменения не привели к появлению ошибки. Обычно для этого используются системы автоматического тестирования. В рамках предлагаемой архитектуры для связи с такими системами выде-

ляются специальные модули интерфейсов. Они работают под управлением тестирующей системы, принимая от нее задания, выполняют нужные действия и передают обратно полученные результаты. На основании сравнения полученного и ожидаемого результата система тестирования может делать вывод о корректной работе тестируемой функции и формировать соответствующие отчеты.

10. Примеры блоков инструментария

На рис. 2 приведена схема инструментария с примерами блоков, расположенных на каждом уровне.

На уровне внутреннего представления располагаются блоки, отвечающие за работу со всеми объектами, составляющими программу: операторами, переменными, типами, комментариями и т.п.

На уровне зависимых от языка блоков находятся блоки, опирающие на специфику входного языка. Например, для определения зависимостей по данным для каждого оператора необходимо знать, какие переменные читаются, а какие изменяются в данном операторе. За это отвечает вот блок «определение используемых переменных». Для нахождения циклов в программе также имеется блок «определение явных циклов». Этот блок перечисляет все записанные явным образом, т.е. с помощью предусмотренных в языке конструкций, циклы, имеющиеся в программе.

На уровне независимых от языка блоков находятся основные блоки анализа и преобразований. Например, здесь располагаются блоки определения информационных зависимостей. Информационную зависимость можно определять различными методами — точным, основанным на графе алгоритма, или разного рода эвристиками. На рисунке показано два блока для этой задачи. Эти блоки имеют одинаковый интерфейс и могут заменять друг друга. На них опирается блок определения свойств параллельности цикла. Здесь тоже два блока с одинаковым интерфейсом, один из которых определяет параллельность цикла на основе наличия в нем зависимостей, а второй получает от эксперта команду, какие циклы считать независимыми, после чего начинает сообщать об этом вызывающим блокам. Это может понадобиться, например, если цикл не поддается анализу, но из

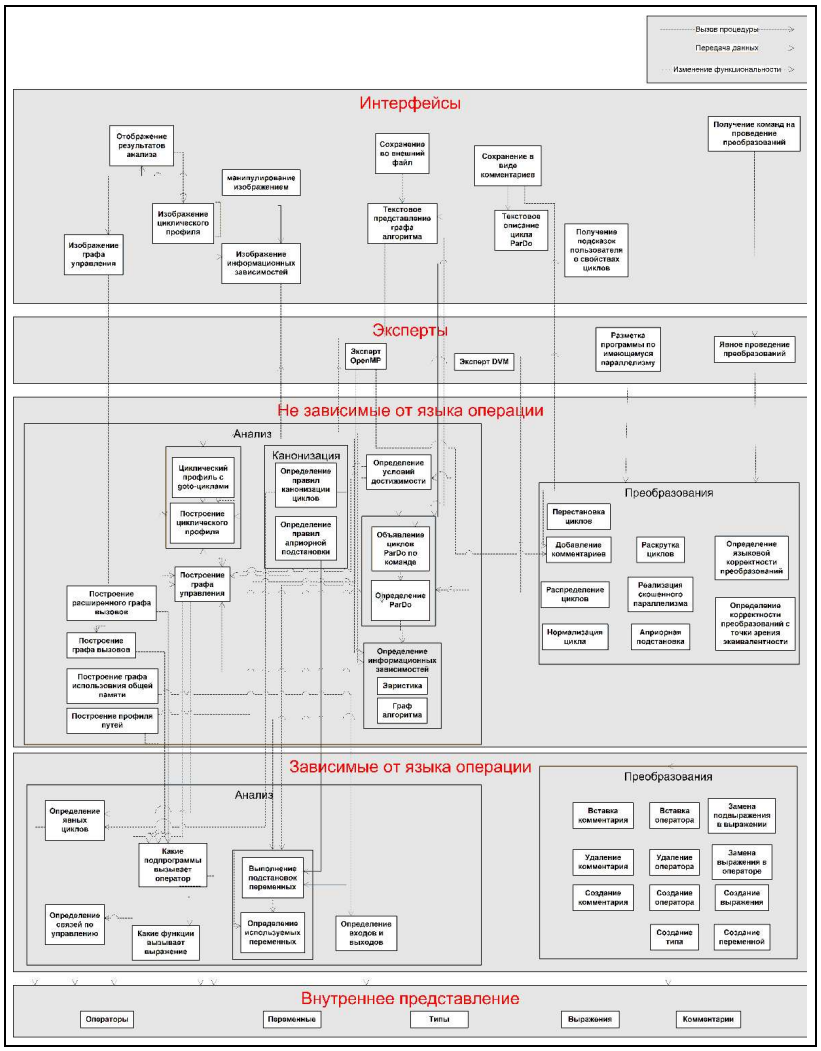

Рис. 2. Логические уровни инструментального комплекса с примерами расположенных на них блоков

каких-то соображений известно, что он параллельный.

В качестве эксперта может быть модуль, реализующий преобразование последовательной программы в систему программирования OpenMP или DVM. Или же конструируемая система (и соответствующий эксперт) может размечать программу в соответствии с имеющимся в ней потенциальным параллелизмом.

На уровне интерфейсов находятся блоки получения представлений по объектам-свойствам («изображение графа управления», «изображение информационных зависимостей»). Также на этом уровне находятся блоки, обеспечивающие взаимодействие с пользователем, в частности, «получение подсказок о свойствах циклов», «получение команд на проведение преобразований».

Список литературы

[1] Shen Z., Li Z., Yew P.-C. An Empirical Study of Fortran Programs for Parallelizing Compilers// IEEE Transactions on Parallel and Distributed Systems. 1990. Vol. 1, no. 3. Pp. 356–364.

[2] Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. — СПб.: БХВ-Петербург, 2002. 608 с.

[3] Хмелев Д.В. Восстановление линейных индексных выражений для сведения программ к линейному классу// ЖВМ и МФ. 1998. Т. 38, № 3. С. 532–544.

[4] Воеводин Вл.В. Теория и практика исследования параллелизма последовательных программ// Программирование. 1992. № 3. С. 38–53.