УДК 519.683+519.688+519.71+519.1

ОСНОВЫ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНОЛОГИИ, ПРЕДНАЗНАЧЕННОЙ ДЛЯ ИССЛЕДОВАНИЯ ПРОГРАММНЫХ СИСТЕМ И ОСНОВАННОЙ НА ПРИНЦИПАХ ЛОГИКО-СМЫСЛОВОГО АНАЛИЗА

Чернышов М. Ю. – к.ф.н., зав. научно-методической частью e-mail: Michael_Yu_Chernyshov@mail.ru (Президиум Иркутского научного центра СО РАН); *Абасов Н. В.* – к.т.н., ведущ. науч. сотрудник, e-mail: cell@sifibr.irk.ru (Институт систем энергетики им. Л. А. Мелентьева СО РАН)

Обсуждается проблема, связанная с отсутствием технологий быстрого создания программных продуктов, обеспечивающих высокую надёжность. Ставится вопрос о необходимости разработки технологии для исследования программ и программных систем в целях повторного использования их или их блоков. Анализируются проблемы, мешающие эффективному анализу готовых программ. Обсуждаются причины необходимости в единых принципах построения программ и подходы к созданию программ на базе таких принципов. Излагаются основы предлагаемой авторами технологии вычислительного моделирования, рассчитанной на эффективный анализ и преобразование программных систем.

The problem bound up with the absence of technologies of rapid development of software products, which could provide for high reliability, is discussed. The need for the development of technologies to study programs and program systems to reuse them or their blocks is considered. Problems, which hamper efficient analysis of software, are discussed. The reasons for the need for universal principles in constructing software and the approaches to elaboration of such principles-based software are discussed. We propose the technology of computational modeling intended for efficient analysis and transformation of the software system whose foundations are outlined.

Ключевые слова: технология быстрого создания программных продуктов, повторное использование программ и блоков программ, эффективный анализ программ, технология вычислительного моделирования.

Введение.

В ходе технолого-коммерческой гонки второго десятилетия XXI века появляются новые высокопроизводительные вычислительные системы. Однако не секрет, что они строятся на старых принципах, т.к. остаются нерешенными

важнейшие задачи, о которых все знают, но предпочитают не говорить. Их нерешенность постоянно сказывается и непременно станет существенным препятствием движению вперед в ближайшем будущем. Общеизвестно, например, что решение задачи создания по большому счёту высокопроизводительных вычислительных систем по-прежнему упирается в нерешенность следующих основных проблем: (1) проблема дефицита эффективных по большому счёту аппаратных (hardware) технологий, т.к. само по себе наличие кластера и удачной вроде бы архитектуры ещё не решает задачу достижения высокой производительности; (2) проблема дефицита вычислительных и программных систем высокой надёжности, что важнее, чем наличие систем высокой производительности; (3) проблема дефицита вычислительных технологий и технологий быстрого создания программных продуктов, обеспечивающих высокую надёжность, что не менее важно. Обсуждая подход к решению последней из проблем, мы коснёмся лишь одной из ключевых задач – задачи разработки технологии для исследования программ и программных систем на принципах смыслового анализа задачи в целях обеспечения условий быстрого и качественного создания новых программных продуктов.

О причинах необходимости в технологии исследования программ и программных систем.

Создание программы сложный и небыстрый процесс. Вот почему всегда существовал искус прибегнуть к повторному использованию уже созданных программ. На начало XXI века в национальном банке алгоритмов и программ (БАП) накоплено огромное количестве программных продуктов. Не меньшее количество алгоритмов и программ можно найти в WWW. Казалось бы, есть возможность взять и воспользовться. Кроме того, любому программисту было бы престижно и выгодно, если бы написанные им программы были использованы в новых разработках, разумеется, при сохранении авторских прав. Но вот – проблема: каждый раз, когда ставится задача создания новой программной системы, программист, естественно, надеется воспользоваться резервами БАП. Однако всякий раз повторяется одна и та же история: программист знает, что алгоритмов масса, но, приступая к решению конкретной задачи программирования разрабатываемой им системы, он вынужден вновь и вновь, как если бы все те базы программ были пусты, каждый раз программировать всё заново, с нуля. Оценки показывают, что так результат достигается быстрее. Почему бы?

Действительно, в БАП имеется огромное количество различных программ, причем, даже с открытым исходным кодом на C, C++, Java, Pascal и др. Любой такой программный продукт «издали» выглядит как вроде бы вполне определенный и понятный, однако при ближайшем рассмотрении, «вблизи», он оказывается совершенно иным. Несмотря на видимость открытости кода, анализ текста таких программ оказывается затруднительным до такой степени, как если бы их исходный код был закрыт. Это объясняется

трудностями эффективного анализа готовых программ в целях их повторного использования.

О проблемах, препятствующих эффективному анализу готовых программ в целях их повторного использования.

Есть несколько объяснений невозможности быстрого и качественного анализа готовой программы: 1) реальная программа содержит тысячи файлов, сотни тысяч строк и десятки тысяч отношений между функциями; 2) неочевидны базовые принципы, на основе которых она построена; 3) последнее предполагает неочевидность базовых механизмов, поддерживающих технологию программирования, использованную при создании этой программы.

К принципиальным недостаткам практически всех программных систем, которые доступны в БАП, следует отнести то, что: (1) они не построены (и сейчас всё ещё не строятся) на единых принципах модельной логико-смысловой организации. Говоря конкретно, программы из БАП (а) имеют различную, причём, часто неочевидную структуру, (б) содержат множество подпрограмм с неочевидной структурой и назначением, и, следовательно, каждая из этих подпрограмм не оказывается органичным и легко вычленимым элементом в модели объектной программной системы. В итоге, анализ этих программ требует очень большого времени, т.к. будучи организованы лишь в структурно-логическом отношении, они, как правило, изначально не организованы в (а) логико-смысловом отношении и (б) интенционально-смысловом (т.е. связанном с их назначением) отношении, а потому очень трудно оценить пригодность объектных программ и их подпрограмм для повторного использования во вновь разрабатываемой программе. Кроме того, (2) не ясно обеспечит ли использование готовых модулей в создаваемой программе высокую эффективность и надёжность программного продукта, если их погружение в новую программу будет осуществлено в соответствии с известными технологиями программирования. Гарантии надёжности функционирования создаваемых программных систем в перспективе требуют обеспечить их *прозрачность*, причём, не только в структурном отношении, а, прежде всего, в логико-смысловом и интенционально-смысловом отношениях. Пока же представление любой программы из БАП по-прежнему ограничено лишь текстовой формой. Отражение даже просто информационно-логического содержания её логико-смысловой модели не предполагается. Между тем, (3) важно обеспечить не одну, а множество форм модельного представления алгоритма программы, которые были бы удобны для аналитика и пользователя, например, а) отображение функциональных логических, отношений и зависимостей в форме графов или сетей, б) отображение логико-смысловых отношений в терминах логических моделей или формул, в) модельное отображение интенционального смысла алгоритма и, как следствие, программы (последнее не оно и то же, что логические отношения внутри модуля или между модулями). Анализ множества таких отображений позволил бы упростить понимание логико-смысловой организации модулей в объ-

ектной программе и, следовательно, оценить их пригодность для целей повторного использования. Практика программирования свидетельствует о такой необходимости. В итоге, (4) пока логико-смысловую модель любой программной системы невозможно оптимально по времени распознать, следовательно, невозможно вычленить из программы полезный элемент. Как следствие, (5) невозможно изменить или полностью перестроить объектную программную систему с учётом возникших потребностей пользователя, а проблемы с адаптацией программного продукта к потребностям пользователя возникают постоянно.

О проблемах, связанных с анализом интенционального смысла программы.

Проблемы, связанные с анализом интенционального смысла программы, заложенного в её алгоритме, очень напоминают проблемы, связанные со сложностями смыслового анализа текстов на естественном языке (ЕЯ) [1]. Если внешняя форма ЕЯ-текста прозрачна, т.е. является отображением ясного, конкретного и логически выстроенного смыслового содержания, то такой текст легко поддается анализу, и его смысл открыт для адресата. Однако часто смысл ЕЯ-текста оказывается неочевидным. Он может быть выражен невербально (экстралингвистическим средствами или средствами подтекста), т.е. не через посредство логико-смысловых отношений текста. Он может быть выражен вербально, но неявно (напр. иносказательно), и тогда его очень трудно выявить путём анализа даже вроде бы очевидной (логически выстроенной) вербальной последовательности текста. По существу, решению задач эффективного анализа и распознавания интенционального смысла частей программы и программы в целом (т.е. их назначения согласно спецификации) препятствуют аналогичные проблемы: 1) проблемы, связанные с неочевидностью логико-смысловых отношений в тексте; 2) проблема, связанная с тем, что старые программы писались на принципах старых технологий программирования (как если бы – старых форм ЕЯ). Вот почему необходимо добиваться того, чтобы все создаваемые программы строились на некоторых единых, универсальных принципах. Но какими должны быть эти принципы?

О возможных единых и универсальных принципах построения программ.

По идее, прозрачная (открытая) внешняя форма программы могла бы, как минимум, отражать информационно-логическое содержание логико-смысловой модели программной системы. Последняя должна быть открытой для аналитика. Уже при выполнении только этого условия программы из БАП читались бы относительно легко. В идеале же, аналитику должны быть доступны (а) логико-смысловая и (б) интенционально-смысловая модели анализируемой программной системы. Они должны открывать для аналитика её логико-функциональное, функционально-смысловое и интенциональное содержание.

Выполненный нами анализ возможных подходов к построению программ на единых, универсальных принципах подсказал нам следующие мысли. *Во-первых*, и это первое, что приходит на ум, возникает мысль о необходимости повышения уровня языка программирования. Но лишь эта мера сама по себе не позволяет решить задачу универсализации основы, т.к. не решается одна из основных задач: программу, написанную даже на языке очень высокого уровня, но на основе традиционных технологий программирования, трудно адаптировать к потребностям пользователя.

Во-вторых, всегда необходимо учитывать, что *понятие о языке программирования* не ограничено лишь формальными правилами записи текста программы в терминах синтаксиса того или иного языка программирования высокого или низкого уровня и соответствующими представлениями о собственно языке, описанном в терминах правил. В результате труда программиста, т.е. в программном продукте, обязательно находят отражение стиль программирования, присущий данному программисту, его умение употреблять или строить сложные операторные конструкции или же обходиться простыми конструкциями, умение формировать и выстраивать последовательность блоков программного продукта, умение или неспособность (нежелание) оптимизировать его и т.д. Таким образом, в языке программного продукта отражаются характер и особенности отношений, складывающихся в системе “программист–программа”. На практике, тексты программ, представляющие собой инструкции для компилятора, действительно аккумулируют в себе особенности и отражают специфические черты, присущие почерку их разработчика. Следовательно, одним из препятствий в деле анализа программ и построения программ на универсальных принципах является *необходимость учёта уникального стиля программирования*, *используемого каждым программистом*.

В-третьих, необходимой фазой в создании программы (и тем более сложной программной системы) является отладка. При отладке огромное время уходит на вычистку деталей, связанных с особенностями и недоработками, которые особенно характерны для западных технологий «сборочного программирования». Такие технологии не предполагают ориентацию на дальнейшую оптимизацию функционирования разработанных программ и, тем более, на усовершенствование методов работы над программами. Между тем, многие отечественные программные комплексы (например, такие, как «Эвролог») содержали в себе весьма перспективные идеи, предполагавшие поиск подходов к усовершенствованию не только программ, но также методов работы над программами, дальнейшее развитие методов и технологий программирования.

В-четвертых, одной из основных фаз жизненного цикла программного продукта является его сопровождение. В связи с этим, разработчикам программного обеспечения часто приходится сталкиваться с потребностями в реинжиниринге, и связано это со следующими объективными факторами:

– на данный момент накоплено большое количество морально устаревшего программного обеспечения, требующего поддержки, модернизации или замены;

– современное программное обеспечение, построенное на принципах «сборочного программирования», характеризуется гигантским объемом исходного кода (тысячи файлов, сотни тысяч строк и десятки тысяч отношений между функциями). Если мы захотели бы воспользоваться чьей-то программой, то столкнулись бы с очевидными трудностями, ведь предварительное исследование столь объёмного исходного кода без использования специализированных инструментальных средств, потребовало бы огромных трудозатрат, большая часть которых ушла бы на выполнение рутинных операций, связанных с поиском и анализом тех или иных связей между блоками программы. Не секрет, что часто такое исследование оказывается невозможным. Игнорирование же должного исследования программной системы при её отладке становится в дальнейшем причиной сбоев. Постоянные сбои в функционировании программных систем (fatal errors and emergency terminations of operation) по причине игнорирования отладки конечного продукта являются практически нормой в работе программ (например, WORD), функционирующих в среде WINDOWS. Иногда такие сбои требуют перезагрузки системы.

В этой связи актуальной представляется следующая постановка задач.

Постановка задач, связанных с переходом к технологии вычислительного моделирования.

По нашему убеждению, задачи создания любых программных систем высокой надёжности должны предполагать соблюдение следующих очевидных требований. 1) Прежде всего, необходимо добиваться того, чтобы создаваемые всеми нами программы строились на неких единых и универсальных принципах модельной логико-смысловой организации.

2) Построение программ и программных систем на принципах модельной логико-смысловой организации предполагает, что следует исходить из соображений о необходимости рассмотрения каждой из подпрограмм как органичного и легко вычленимого логико-смыслового элемента в единой логико-смысловой модели программной системы.

3) Важно, чтобы логико-смысловая модель любой программной системы могла быть эффективно (оптимально по времени) построена, легко изменена или полностью перестроена с учётом возникших потребностей пользователя или задач разработчика системы. Предлагаемый нами подход к решению задачи облегчения анализа программ или программных систем предполагает следование такой технологии программирования, при которой внешняя форма программы оказывается отражением информационно-логического содержания соответствующей логико-смысловой модели этой программы или программной системы. При таком подходе проблемы с адаптацией программного продукта к потребностям пользователя не будут возникать.

4) Для решения поставленных задач необходимо перейти от устаревшей технологии программирования к технологии вычислительного (программирующего) моделирования, т.е. к технологии, предполагающей построение логико-смысловых моделей, как альтернативы технологии, основа которой состоит в примитивном использовании формальных языков и соблюдении норм синтаксической организации сегментов программ. На первом этапе возможный подход к созданию такой технологии может быть ограничен стратегией, предполагающей смысловой анализ задачи, агрегирование логики (так, чтобы любой алгоритм можно было логически четко представить в его развитии от исходной идеи до реализации в деталях) и построение моделей программ на этой основе.

Основы и принципы технологии вычислительного моделирования программ. Обозначим принципиальные основы технологии вычислительного моделирования. В их число входят: (1) обязательное построение интегральной смысловой модели создаваемой или анализируемой программы; (2) разработка подхода к решению задачи по созданию такой модели, т.е. выполнению моделирующего отображения; 3) поиск средств представления (отображения) модели; 4) поиск способов упрощения модели.

(5) Одним из наиболее эффективных подходов к решению этой задачи, предшествующей анализу текста программы, может предполагать моделирующее отображение последнего. Но одно дело – моделирование текстов простых программ. Совсем другое дело – моделирование текстов больших программных систем, состоящих из сотен и тысяч файлов. Вот почему необходим способ упрощения модели.

(6) Моделирующее отображение программы должно (и может) передавать не только её общую структуру, но также её функционально-смысловую структуру, т.е. структуру отношений между её функциями, несущими определенную смысловую нагрузку (Рис. 1).

Если не удаётся выявить структуру отношений между всеми функциями программной системы, т.к. она велика, то необходимо предварительно упростить анализ отношений. Этого можно добиться путём декомпозиции её модели на функционально-смысловые (ФС) блоки, по уровням: ФС-структура программы – ФС-структура на уровне модулей – ФС-структура на уровне операторов.

Если учесть, что в основе операций с множеством отношений в анализируемой программе являются операции, основанные на принципах анализа и синтеза ФС-блоков, то при необходимости ФС-блоки модели легко интегрируются обратно, в единое целое.

(7) Что касается требований к способам моделирующего отображения алгоритмов программ, необходимо обеспечить многообразные отображения одной и той же модели. Это могут быть, например, отображения в форме текстов, отображения в форме графов или сетей отношений, отображения логических отношений в терминах логических формул, а также, возможно, ото-

бражения семантических отношений, не совпадающих с логическими отношениями.

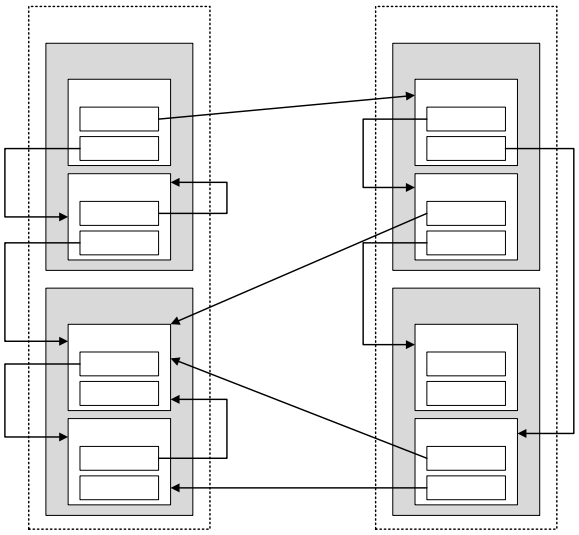

Рис. 1. Схема отношений между функциями в модели анализируемой системы

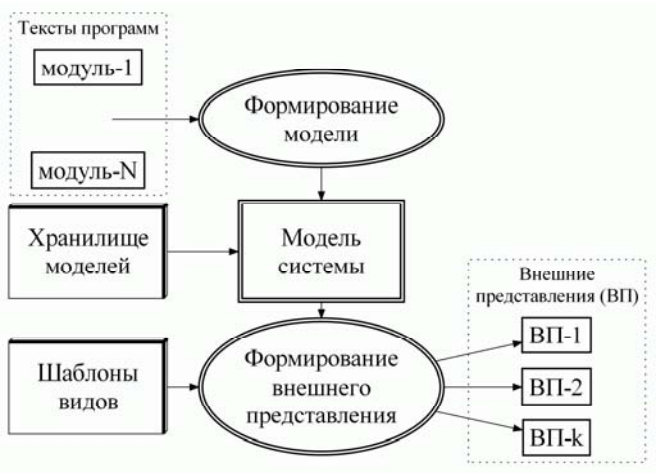

Рис. 2. Условное представление технологии вычислительного моделирования

(8) Одним из наиболее эффективных способов моделирующего отображения алгоритма программы может быть её отображение, предполагающее формирование графа (или иерархии графов) функционально-смысловых отношений с различными уровнями представления внутренних объектов и связей.

Заключение. О воплощении вычислительной технологии для исследования программных систем.

Существует множество подходов к решению поставленной нами задачи, связанной с разработкой технологии вычислительного моделирования. Мы выбрали тот, который предполагает создание практически работающей системы и практическое опробование на ней возможности эффективного исследования объектных систем. Особенно сложно реализовать такой подход в случае, если разрабатываемая система должна реализовывать функции логико-смыслового анализа. И всё же авторы избрали именно этот путь. На основах, описанных выше, удалось построить эффективную технологию и воплотить её в форме аналитического программного комплекса. Инструментальными основами технологии стали (1) разработанная нами универсальная инструментальная среда программирования ЗИРУС [2] и (2) подход, предполагавший представление компьютерных программ в виде моделей данных и набора отношений [3]. Он был основан на общих принципах теории баз данных, применении теории графов в качестве аппарата модельного отображения программной системы/модуля [4] и рассчитан на формирование различных внешних текстовых и графических модельных отображений программ. Существенный сдвиг вперёд был обеспечен за счёт решения, связанного с упрощением представления сложных алгоритмов больших программ. На этой основе, в рамках проекта по формированию графов функциональных связей анализируемой программы по её исходному коду, авторами сначала была разработана вычислительная аналитическая система, предназначенная для автоматизированного логико-смыслового анализа обширного класса модулей программ и программных систем в целях изучения возможности их повторного использования во вновь создаваемых программных продуктах. Логическая основа этой аналитической системы была написана на языках Lua и Python. Система предполагает решение задачи статического анализа кода анализируемой программы с помощью оригинальной технологии автоматического построения графов связей в её тексте, причём, анализ производится без выполнения программы, над версией исходного кода. Глубина анализа может варьировать от определения поведения отдельных операторов программы до анализа всего исходного кода. Программный комплекс уже нашел практическое применение в решении задач оперативного выявления мест в объектных программах, содержащих ошибки, идентификации свойств программ, их соответствия спецификациям и т.д.

Затем, на основе этой аналитической системы, была разработана вычислительная система, предназначенная уже для исследования и модификации программных систем с чётом итогов их логико-смыслового анализа [4]. Эта версия предлагает новую возможность: модификация программы в соответствии с возникшими потребностями. Теперь в процессе анализа можно применять предустановленные шаблоны различных представлений графов, а

также дополнять графы новыми кластерами, созданными аналитиком-пользователем или программистом-пользователем.

Библиографические ссылки

1. *Чернышов М. Ю.* Проблема идентификации вербально выраженных мыслей-скрепов как медиаторов смысловой интегративности текстов / М. Ю. Чернышов. М.: Наука и право, 2010. 152 с.

2. *Абасов Н. В.* Основы универсальной среды программирования ЗИРУС / Н. В. Абасов // Вестник ИрГТУ. - 2006. - № 2(26). - С. 62–68.

3. *Абасов Н. В.*, *Чернышов М. Ю.* На пути к вычислительной технологии для эффективного исследования программ и программных систем, основанной на принципах логико-смыслового анализа / Н. В. Абасов, М. Ю. Чернышов // Информационные технологии и высокопроизводительные вычисления. Материалы международной конференции (Хабаровск, 4–6 октября 2011 г.). Хабаровск: Изд-во ТОГУ, 2011. С. 3–15.

4. *Абасов Н. В., Миронов В. О.* Применение графов для анализа и реструктуризации исходных текстов программ / Н. В. Абасов, В. О. Миронов // Труды XIV Всероссийской Байкальской конф. “Информационные и математические технологии в науке и управлении”. Т.3. Иркутск: ИСЭМ СО РАН, 2009. С. 173–178.