11. Goranskii V. P. WinEfk software : [web]. [2009]. Available at: http://www.variablestars.ru/FILES/win-efk.rar.

12. Palaversa L., Ivezic Z., Eyer L. et al. 2013, arXiv: 1308.0357 [astro-ph.GA].

13. Catalina Sky Survey : Cone Search Service . Available at: ttp://nesssi.cacr.caltech.edu/cgi-bin/getcssconedb_release.cgi.

УДК 004.42

ОСОБЕННОСТИ ХРАНЕНИЯ ФУНКЦИОНАЛЬНО-ПОТОКОВЫХ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ*

А. И. Легалов[1], И. В. Матковский[2], А. В. Анкудинов[1]

[1]Сибирский государственный аэрокосмический университет имени академика М. Ф. Решетнева
Российская Федерация, 660014, Красноярск, просп. им. газ. «Красноярский рабочий», 31.
E-mail: legalov@mail.ru
[2]Сибирский федеральный университет
Российская Федерация, 660041, Красноярск, просп. Свободный, 79. E-mail: alpha900i@mail.ru

Представлены особенности хранилища (репозитория) функционально-потоковых параллельных программ, обеспечивающего создание распределенных библиотек функций и совместную разработку программного обеспечения. Показана специфика репозитория, его место в системе функционально-потокового параллельного программирования. Описывается логическая структура хранилища и ее взаимосвязь с языком программирования. Применение репозитория обеспечивает поддержку эволюционного расширения программ без изменения ранее написанного кода и может использоваться для хранения не только функций, но и других программных объектов.

Ключевые слова: хранение программ, репозиторий, функционально-потоковое параллельное программирование, разработка программного обеспечения, расширение программ, инструменты для разработки программного обеспечения.

PECULIARITIES OF THE FUNCTIONAL DATAFLOW PARALLEL PROGRAM STORAGE

A. I. Legalov[1], I. V. Matkovsky[2], A. V. Ankudinov[1]

[1]Siberian State Aerospace University named after academician M. F. Reshetnev
31, Krasnoyarsky Rabochy Av., Krasnoyarsk, 660014, Russian Federation. E-mail: legalov@mail.ru
[2]Siberian Federal University
69, Svobodny Av., Krasnoyarsk, 660041, Russian Federation. E-mail: alpha900i@mail.ru

The peculiarities of the functional dataflow parallel program repository are presented. This repository makes it possible to use the distributed libraries of functions and support collective software development. The specificity of the repository and its place in the system of the functional dataflow parallel programming are shown. The logical structure and relationship with the programming language of repository are described. The usage of the repository allows to support the evolutionary expansion of the programs without changing of the early written code and can be used to store not only functions, but other software objects.

Keywords: repository, functional dataflow parallel programming, software development, program extension, software toolkits.

В настоящее время большинство языков программирования ориентируются на текстовое представление программ. При этом программа обычно размещается в нескольких файлах, каждый из которых может нести различную ролевую нагрузку, во многом определяемую спецификой построения конкретного языка программирования. Внутренняя структура этих файлов также бывает различной и зависит как от специфики языка, так и от особенностей использования в нем того или иного файла. Из отдельных единиц компиляции программа может собираться в единый исполняемый модуль с применением различных инструментальных средств. На разных стадиях этого процесса используется препроцессорная обработка, ком-

* Работа поддержана грантом в рамках РФФИ № 13-01-00360 «Методы и средства эволюционной разработки программного обеспечения с применением процедурно-параметрической парадигмы программирования».

пиляция, статическая и динамическая компоновка, а также ручная сборка.

Решая конкретную задачу, программисту зачастую приходится манипулировать дополнительными понятиями, которые не несут семантически полезной нагрузки, связанной с предметной областью. Это обусловливается особенностями общей структуры программы, определяемой спецификой используемых инструментальных средств, а также необходимостью поддержки разнообразных критериев качества. В частности, раздельная компиляция удобна для написания расширяемых программ и позволяет модифицировать отдельные фрагменты кода, не затрагивая прочие единицы компиляции. Но для того, чтобы обеспечить повышение эффективности использования единиц компиляции, приходится применять дополнительные инструменты, такие как программы сборки проекта.

Существует тенденция организации процесса разработки программ таким образом, чтобы разработчик как можно меньше был связан с использованием промежуточных конструктивных понятий, а в большей степени ориентировался на применение тех сущностей, которые бы непосредственно определяли структуру программы. Для этого используются различные подходы, среди которых можно отметить:

– создание для уже разработанных языков программирования инструментальных средств, позволяющих скрыть служебные (вспомогательные) структуры и сервисы;

– разработка новых языков программирования, структура которых изначально старается избавиться от вспомогательных понятий.

Первый подход широко применяется на практике почти со всеми существующими языками. Он ориентирован на интегрированные среды разработки, отображающие только требуемые структуры. Примерами подобных сред являются MS Visual Studio [1], Qt Creator [2] и др. К достоинствам подхода можно отнести использование существующих языков программирования в новом окружении. Недостатки заключаются в том, что среда разработки становится громоздкой из-за поддержки как устаревших, так и новых методов создания программ. Старые концепции постоянно проявляются в различных аспектах разработки программного обеспечения.

Разработка новых языков программирования позволяет избавиться от использования «избыточных» технологических конструктивов. В этом случае с базовой файловой структурой можно непосредственно увязать «полезный» программный объект, например, класс или модуль. В связи с этим инструментальная поддержка процесса выполнения программ выстраивается вокруг только основных понятий языка программирования, используемых для описания предметной области. К достоинствам подхода следует отнести отсутствие промежуточных (вспомогательных) конструктивов, усложняющих лишними проектными решениями программу. Также отпадает необходимость в дополнительных инструментальных средствах. Среди недостатков можно отметить потерю гибкости при разработке программ с учетом требуемых критериев качества. В частности, в ряде случаев возникают проблемы с эволюционным расширением программ. Для решения этих проблем часто требуется модификация изначально разработанного языка программирования путем включения в него вспомогательных конструкций, обеспечивающих достижение требуемых критериев качества в ходе разработки ПО. Примером такого развития языка является C# [3], в который добавлены возможности по расширению классов в различных файлах.

Таким образом, независимо от того, как развивается язык программирования, возможность использования только полезных конструктивов достигается за счет дополнительных инструментальных средств, таких как интегрированные среды разработки. Подобное решение используется и при расширении функциональных возможностей инструментальных средств, обеспечивающих программирование на языке «Пифагор» [4].

Целью работы является разработка репозитория, обеспечивающего хранение исходных текстов функций, а также различных внутренних представлений, формируемых в ходе создания на языке «Пифагор» функционально потоковых параллельных программ.

Специфика языка «Пифагор». Основной особенностью языка, влияющей на хранение в репозитории отдельных сущностей, является организация имен функций и трактовка (интерпретация, восприятие) этих имен с точки зрения общего пространства имен. Понимание пространства имен в языке «Пифагор» отличается от его трактовки в таких языках программирования, как C++ или C#. В языке отсутствует непосредственное описание пространств имен с использованием тех или иных аналогов описателя *namespace*. Вместо этого допускается написание составного имени функции, в котором отдельные идентификаторы отделяются друг от друга точкой. Самый последний идентификатор в этой цепочке имен определяет имя функции. Все предыдущие идентификаторы воспринимаются как подпространства имен, вложенные друг в друга. Это позволяет гибко расширять пространства имен, добавляя в них новые понятия (функции и константы) без изменения ранее написанных файлов, являющихся единицами хранения функций и констант.

В качестве примера рассмотрим функцию нахождения абсолютной величины числа, размещенную в пространстве имен Math:

```
Math.Abs << funcdef X {
    val << ({X:-},X);
    key << ((X,0):[<,>=]):?;
    return << val: key:.; }
```

Функция может быть записана в любом файле в сочетании с функциями, имеющими префиксы других пространств имен. Например, функция, осуществляющая нахождение скалярного произведения двух векторов, может находиться в пространстве имен Vector:

```
Vector.VVMult << funcdef Param {
    (Param:#:[]:*):VSum:return };
```

Функция может не содержать префиксов пространств имен:

```
// Функция, возвращающая произведение матрицы на скаляр
MSMult << funcdef Param {
    A << Param:1;
    (A,(Param:2,A:|):dup):#:[]:VSMult:(.):return  };
```

В этом случае предполагается, что она хранится во внешнем (безымянном) пространстве, расположенном на самой вершине иерархии пространств имен.

Навигация по такой системе функций, составляющих единую библиотечную среду, становится неудобной. Необходимо таким образом организовать их хранение, чтобы иерархия пространств имен, формируемая децентрализовано, составляла в результате единое логическое и физическое пространство, доступное программисту.

Логическая структура репозитория. В соответствии с особенностями языка функционально-потокового параллельного программирования «Пифагор», логическая структура репозитория, предназначенного для хранения функций, должна представлять иерархическую структуру, состоящую из вложенных подпространств имен, исходные тексты функций и констант, их промежуточные представления и прочие данные. Описание данной структуры можно выразить с использованием следующих правил:

Репозиторий = Заголовок БезымянноеПространствоИмен.
Заголовок = ИмяРепозитория.
БезымянноеПространствоИмен =
[Функции] [ИменованныеПространства] [Константы].
Функции = Функция {Функция}.
Константы = Константа {Константа}.
ИменованныеПространства =
ИменованноеПространство { ИменованноеПространство}.
Функция = ИмяФункции ИсходныйТекстФункции
ПромежуточныеПредставленияФункции.
Константа = ИмяКонстанты ИсходныйТекстКонстанты
ПромежуточныеПредставленияКонстанты.
ИменованноеПространство = Заголовок
Функции [ИменованныеПространства] [Константы] |
[Функции] ИменованныеПространства [Константы] |
[Функции] [ИменованныеПространства] Константы].
Имя… = Идентификатор.

В качестве примера можно рассмотреть формирование репозитория для функций, приведенных выше. Пусть будет создан пользовательский репозиторий с именем ***MyRepository***. Тогда хранимые в нем данные можно представить в виде иерархии:

MyRepository
MSMult
-- Исходный текст функции MSMult
Math
Abs
-- Исходный текст функции Math.Abs.
Fact
-- Исходный текст функции Math.Fact
Vector
VVMult
-- Исходный текст функции Vector.VVMult

Физическая структура репозитория. Иерархическая структура хранилища, обеспечиваемая репозиторием, может быть организована с применением различных подходов. В качестве возможных вариантов можно выделить использование файловой системы или систему управления базами данных (СУБД). Организация репозитория хорошо ложится на структуры современных файловых систем. Это позволяет обеспечить достаточно быструю реализацию основных функций, опираясь только на системные вызовы используемых операционных систем. Применение переносимых библиотек для организации работы с файловыми системами, таких как boost, Qt и др., позволяет создавать программное обеспечение, обеспечивающее функционирование репозиториев на различных операционных системах. Аналог подобного решения применен в ранее реализованных проектах с использованием библиотеки Qt. Вместе с тем следует отметить, что реализация репозитория на основе файловой системы затрудняет добавление дополнительных функций, таких как поиск по репозиторию, навигация, фильтрация расположенных в нем данных, а также замедляет их работу.

Реализация вспомогательных функций более эффективно осуществляется при использовании СУБД. В этом случае применяется дополнительная инструментальная поддержка работы с данными, что ведет к повышению эффективности процесса разработки. К преимуществам данного подхода можно отнести использование различных вариантов архитектуры «клиент–сервер». Его недостатками являются использование более сложной инструментальной поддержки и необходимость дополнительной реализации системы поддержки версий в случае коллективной работы над репозиторием.

Инструментальная поддержка репозитория. Среди основных требований к репозиторию можно выделить:

– децентрализованное хранение данных, при котором каждый из разработчиков может создавать свой репозиторий;

– поддержка коллективной разработки библиотек и версий отдельных функций;

– автоматический учет зависимостей выбранной функции;

– удаленный доступ;

– разграничение прав доступа к отдельным элементам репозитория и ко всему в целом;

– осуществление поиска внутри репозитория;

– возможность полного или частичного исполнения выбранной функции на стороне репозитория;

– графический интерфейс пользователя.

Приняв во внимание описанные выше требования, а также взаимосвязь данного модуля с другими подсистемами разрабатываемого программного комплекса, можно прийти к заключению о необходимости создания клиент-серверного программного обеспечения класса хранилища данных, поддерживающего распределенную модель хранения данных. При этом процесс взаимодействия пользователя репозиторием может быть описан с помощью следующих обобщенных шагов:

1) запускается клиентская часть системы;

2) с помощью средств разработки пишется разрабатываемая функция;

3) при необходимости проводится сохранение написанного исходного кода в заданном репозитории;

4) после этого осуществляется запись функции в репозиторий. При этом будет проведен синтаксический анализ, проанализированы зависимости между создаваемой и вызываемыми функциями, а результаты анализа будут записаны в таблицу зависимости функций.

Проанализировав и приняв во внимание положительные и отрицательные стороны описанных выше вариантов организации репозиториев, можно прийти к выводу о необходимости реализовать клиент-серверное программное обеспечение в виде хранилища данных с базовым набором функциональных возможностей, упомянутых ранее, и возможностью их дальнейшего наращивания. При этом на начальном этапе имеет смысл разработать и реализовать работу с локальным репозиторием, что позволит отработать базовые алгоритмы сборки программы.

Варианты использования репозитория. Доступ к используемым репозиториям осуществляется в зависимости от взаимного расположения библиотечных функций и функций, разрабатываемых программистом. Если разрабатываемые функции расположены в том же репозитории, что и используемые библиотечные функции, то специального указания о подключении репозиториев не требуется. В этом случае вновь создаваемая функция напрямую обращается к используемой библиотечной функции с указанием полного имени в соответствии с ее пространством имен.

Например, если библиотечная функция вычисления квадратного корня *sqrt* размещается в текущем репозитории в пространстве имен *math.elementary*, то доступ к ней осуществляется по имени *math.elementary.sqrt*. При разработке функции вычисления диагонали квадрата по заданной стороне, расположенной в том же хранилище, но в пространстве имен *user.figures.square*, получим следующее решение:

```
user.figures.square.diagonal <<
funcdef a {
      (a,a):* >> aa;
      (aa,aa):+:math.elementary.sqrt >>
return; }
```

В том случае, если библиотечный репозиторий размещен в другом месте, то для доступа к нему используется оператор импорта. При этом местоположение репозитория указывается с применением широко используемого механизма URL (Universal Resource Locator). Пусть библиотечный репозиторий располагается в отдельном каталоге пользователя, например, */home/username/lib-repo/* на этом же компьютере, что разрабатываемый код. Тогда разрабатываемая функция будет выглядеть следующим образом:

```
sqrt << import
"file://home/usename/lib-
repo/math.elementary.sqrt";
user.figures.square.diagonal <<
funcdef a {
      (a,a):* >> aa;
      (aa,aa):+:sqrt >> return; }
```

Возможна также ситуация, когда в разрабатываемой функции используются несколько функций из библиотечного репозитория. В этом случае можно указать только путь к нему, подставив вместо подпространств имен и имен библиотечных функций звездочку «*». Приведенная в качестве примера функция будет выглядеть следующим образом:

```
ext_repository << import
"file://home/usename/lib-repo/*";
user.figures.square.diagonal <<
funcdef a {
      (a,a):* >> aa;
      (aa,aa):+:
ext_repository.math.elementary.sqrt >>
return; }
```

В качестве URL может быть указан и удаленный репозиторий, расположенный, например, на ftp- или http-сервере. Тогда в операторе импорта задается соответствующий URL с указанием сервера. Например:

```
ftp_repository << import
"ftp://ftp_srever_name/*";
http_repository << import
"http://http_server_name/*";
```

Интеграция языковых и инструментальных средств повышает эффективность процесса разработки программного обеспечения, предоставляет в распоряжение программиста новые возможности по сопровождению и расширению программ, скрывая при этом многие вспомогательные операции, напрямую не связанные с решаемой прикладной задачей. Предлагаемый репозиторий, обеспечивающий распределенное хранение функционально-потоковых параллельных программ, позволяет повысить эффективность совместного использования разрабатываемых программ, накопление и обмен создаваемыми библиотечными функциями. Помимо этого предлагаемые методы могут также использоваться при разработке других иерархических распределенных хранилищ, например, при расширении обобщенных записей и обобщающих процедур в процедурно-параметрическом программировании [5]. Ряд функций, обеспечивающих распределенное хранение данных, также использован в системе многокритериального анализа [6].

Библиографические ссылки

1. Хортон А. Visual C++ 2005: базовый курс : пер. с англ. М. : Вильямс, 2007. 1152 с.

2. Шлее М. Qt 4.8. Профессиональное программирования на C++. СПб. : БХВ-Петербург, 2012. 894 с.

3. Троелсен Э. С. C# и платформа .NET. Библиотека программиста : пер. с англ. СПб. : Питер, 2003. 800 с.

4. Легалов А. И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. 2005. № 1 (10). С. 71–89.

5. Легалов А. И., Солоха А. Ф. Особенности языка процедурно-параметрического программирования // Вестник Новосибирского гос. ун-та. Сер. Информационные технологии. 2011. Т. 9, № 3. С. 15–22.

6. Легалов А. И., Ледяев Д. Н., Анкудинов А. В. Инструментальная поддержка многокритериального анализа при разработке сложных технических систем // Вестник СибГАУ. 2009. Вып. 2 (23). С. 50–55.

References

1. Horton I. *Visual C++ 2005: bazoviy kurs* [Beginning Visual C++ 2005]. Moscow, I. D. Williams publ., 2007, 1152 p.

2. Schlee M. *Qt 4.8. Qt 4.8. Professionanoel programmirovanie na C++* [Qt 4.8. C++ Professional Programming]. Saint-Petersburg, BHW-Petersburg, 2012, 894 p.

3. Troelsen A. *C# i platforma .NET* [C# and the .NET Platform]. Sanct-Petersburg, Piter, 2003, 800 p.

4. Legalov A. I. *Vychislytel'nye technologii.* 2005, № 1 (10), p. 71–89.

5. Legalov A. I., Soloha A. F. *Vestnik NGU*, Information technologies. 2011, vol. 9, № 3, p. 15–22.

6. Legalov A. I., Ledyaev D. N., Ankudinov A. V. *Vestnik SibGAU*. 2009, no. 2 (23), p. 50–55.

УДК 62.501

О НЕКОТОРЫХ ИССЛЕДОВАНИЯХ ПО СИСТЕМНОМУ АНАЛИЗУ

А. В. Медведев

Сибирский государственный аэрокосмический университет имени академика М. Ф. Решетнева
Российская Федерация, Красноярск, просп. им. газ. «Красноярский рабочий», 31
E-mail: Saor_medvedev@sibsau.ru

Рассматриваются некоторые аспекты системного анализа и теории исследования операций. Обсуждаются также наиболее важные основные черты последних. Приводится конкретный пример эффективного применения системного анализа в задаче управления техническим объектом. Анализируются некоторые вопросы информатизации управленческих решений для различных процессов, включая организационные. Описаны некоторые принципиальные особенности создания систем моделирования и управления активными объектами. Выделены наиболее важные задачи системного анализа. Дается краткая характеристика исследований в области системного анализа и исследования операций, проводимых в течение последних лет на кафедре системного анализа и исследования операций СибГАУ.

Ключевые слова: активные системы, сложные процессы, априорная информация, критерии, ограничения, обучающиеся модели, алгоритмы оптимизации, принятие решений.

ABOUT SYSTEM ANALISIS AND OPERATION RESEARCH THEORY

A. V. Medvedev

Siberian State Aerospace University named after academician M. F. Reshetnev
31, Krasnoyarsky Rabochy Av., Krasnoyarsk, 660014, Russian Federation. E-mail: Saor_medvedev@sibsau.ru