Дрождин В.В., Жуков М.В. Экспертно – ориентированный подход к разработке прикладного программного обеспечения. // Проблемы информатики в образовании, управлении, экономике и технике: Сб. статей IX Междунар. научно-техн. конф. – Пенза: ПДЗ, 2009. – С. 131-135.

ЭКСПЕРТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД К РАЗРАБОТКЕ ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В.В. Дрождин, М.В. Жуков

Пензенский государственный педагогический университет
им. В.Г. Белинского,
г. Пенза, Россия

Разработка программного обеспечения (ПО) для любой предметной области обязательно требует привлечения эксперта и программиста. Целью исследования является разработка метода формирования программ, позволяющего исключить программиста из этого процесса. Существует несколько способов достижения такого результата. Например, обучить эксперта навыкам программирования или предоставить эксперту средства формальной спецификации предметной области и решения задач, позволяющие разрабатывать предметно-ориентированное ПО, ограничиваясь только знаниями предметной области. В статье приводятся требования к инструментальным средствам создания ПО без участия программиста, рассмотрены вопросы конструирования, автоматического синтеза и оптимизации программ с учетом специфики предметной области.

Drozhdin V.V., Zhukov M.V. Expert-oriented approach to the development of applied software.

Both programmer and expert are needed for software development in any knowledge domain. The purpose of this research is to develop the methods of programs' construction without programmers. There are several approaches to do this. One of them is to teach the expert to write a program. The other approach is to provide the expert with both formal specification and problem solving tools, allowing to develop object-oriented software with domain knowledge. The article covers the claims for tools of software creation without programmers and the questions of construction, automatic synthesis and software optimization by the domain specific.

Конечно, фраза «программирование без программистов» звучит парадоксально, однако сведение роли программиста к минимуму в процессе разработки прикладного ПО вполне реально.

За основу взят подход, предложенный в [1]: группа экспертов совместно с программистами создают библиотеку компонентов, содержащую решения типовых задач (аксиомы) автоматизируемой предметной области и разрабатывают средства, позволяющие решать специфические задачи, возникающие перед экспертами. Решение каждой такой задачи представляет собой композицию решений типовых задач. Таким образом, библиотека должна обладать свойством полноты, позволяющим сводить любую задачу, возникающую в предметной области, к решению типовых задач. Данный подход не нов и наиболее успешный пример его реализации электронная таблица MS Excel.

1. Библиотека компонентов. Задача создания библиотеки компонентов, обладающей свойством полноты, сводится к всестороннему анализу предметной

области (Domain Analysis). Наиболее успешно зарекомендовал себя подход FODA (Feature Oriented Domain Analysis) описанный в [2].

Примечание: в качестве альтернативы тандема «эксперт + программист» можно использовать расширяемые on-line библиотеки, в которые каждый желающий, соблюдая определенные требования, сможет добавлять свои компоненты.

2. Композиции компонентов. Построение композиции элементов осуществляется на языке CDL (Composition Description Language). Язык содержит операторы обращения к компонентам, средства сохранения промежуточных результатов, условные и циклические операторы. Кроме этого, язык содержит набор операторов трансформации компонентов, используемых для оптимизации компонентов. На основе CDL-программы происходит генерация исполняемого кода компонента на GPL (General Purpose Language).

Язык GPL понятен программисту, но не понятен неискушенному пользователю, поэтому перед экспертами и программистами возникает еще одна совместная задача: разработка DSL (Domain Specific Language). Цель языка DSL – сделать максимально понятной и наглядной для экспертов процедуру решения специфических задач. В качестве DSL могут выступать визуальные средства, как это сделано в MS Excel, описания на естественном языке (такой подход используется в экспертной системе DARPA's Rapid Knowledge Formation (RKF) project [3]). Описание компонента на DSL преобразуется в CDL.

DSL – язык, ориентированный на формальное описание предметной области [4].

3. Оптимизация и автоматический синтез компонентов. Можно ли сделать библиотеку компонентом активной, способной самостоятельно улучшать качество находящихся в ней компонентов? Можно ли разработать средства автоматической генерации компонентов в условиях, когда эксперт знает то, что он хочет получить, но не знает, как этого достичь? Прежде чем ответить на эти вопросы, давайте разберемся, как человек решает поставленные перед ним задачи и как он объясняет компьютеру, как решить ту или иную задачу.

Семантика. Рассмотрим весьма типичную ситуацию: жаркий июньский полдень, раскаленный воздух и человека, идущего по улице, у которого возникает желание утолить жажду. Каковы будут его действия? Очевидно, что первым делом он построит план, как удовлетворить свое желание: найти ближайший магазин, купить воду, выпить ее, тем самым, утолив жажду. Затем, он взвесит, есть ли у него необходимые ресурсы: время, деньги и только после этого начнет исполнять намеченный план. Таким образом, одного только желания достаточно для человека, чтобы построить план действий, оценить имеющиеся ресурсы и выполнить необходимую последовательность действий для его осуществления.

А теперь давайте представим, что нам нужно заставить компьютер сделать то же самое, т.е. запрограммировать решение некоторой задачи. Насколько это сложнее? Чтобы ответить на этот вопрос разберемся с семантикой тех синтаксических единиц, с помощью которых происходит общение «Человек-Человек» (хотя в приведенном примере и рассматривается взаимодействие «Человека» с самим собой оно может быть сведено к связи «Человек-Человек» с помощью просьбы «Дай мне попить») и «Человек-Компьютер».

Анализ начнем с взаимодействия «Человек-Человек». Когда мы детально описываем последовательность действий, которую необходимо осуществить человеку для утоления своей жажды, мы задаем операционную семантику. В нашем случае ее можно представить в виде: «сделать 40 шагов до магазина → подойти к прилавку → взять воду → подойти к кассе → рассчитаться → выпить воду». Если мы, например, после действия «взять воду» добавим действие «проверить, что вода холодная», то мы получим совершенно другую операционную семантику.

Операционная семантика – последовательность шагов, приводящая от исходных данных к конечному результату [4].

Составляя план действий, мы переходим на уровень денотационной семантики. На этом уровне нас не интересуют последовательность шагов, которую нужно осуществить для достижения конечного результата, нас интересует лишь конечный результат – реализация поставленных целей: «утолить жажду (купить воду, выпить воду)».

Денотационная семантика – задается путем формальных функций, отображающих входные данные в конечный результат. Здесь не делается акцент на процедуре достижения конечного результата; здесь лишь интересна зависимость между входными данными и некоторым результатом [4].

Последний вид семантики, который нам осталось рассмотреть – аксиоматическая семантика. Наличие желания утолить жажду денег и свободного времени – это аксиоматическая семантика входа; жажда утолена – выхода. Или более формально в виде тройки Хоара {«желание утолить жажду» ^ «есть деньги» ^ «есть свободное время»} ? {«жажда утолена»}.

Аксиоматическая семантика – утверждения о свойствах входных и выходных данных [4].

Теперь рассмотрим, как происходит взаимодействие «Человек-Компьютер». Мы оперируем операционной семантикой, когда пишем программный код на языках общего назначения (GPL); мы переходим на уровень денотационной семантики, когда используем объектный, функциональный или компонентный подходы; мы оперируем аксиоматической семантикой, когда используем логические языки программирования.

Таким образом, мы не будем изобретать колесо с нуля, а соберем его из имеющихся частей: разработаем библиотеки способные учитывать и оперировать каждым из рассмотренных видов семантик. Так как, только обладая семантическими знаниями, мы способны выполнять оптимизацию с учетом специфики предметной области и автоматически синтезировать программы.

Оптимизация композиций компонентов. Использование компонентного подхода позволит нам проводить оптимизацию на уровне денотационной семантики. Кроме того, как отмечалось в п.2, каждый компонент описывается на DSL, что позволяет проводить оптимизацию с учетом специфики предметной области.

Оптимизация происходит одним из двух способов:

– метод эквивалентных замен компонентов или их композиций. Наличие денотационной и аксиоматической семантики позволяет существенно упростить процедуру поиска эквивалентов: 1) по соотношению слов формального описания, 2) по равенству аксиоматических семантик (вхождение множеств);

– метод отбрасывания редуцированных элементов на основе набора статистики.

Оптимизация вычислительного процесса. Наличие аксиоматической семантики позволяет:

– до окончания вычислительного процесса описать свойства выходных данных;

– до начала вычисления определить достаточно ли информации и ресурсов для его нормального выполнения.

Автоматический синтез компонентов. Наличие аксиоматической семантики позволяет осуществлять дедуктивный синтез компонентов. При таком подходе аксиоматическая семантика, задающая соотношение между входными и выходными данными, рассматривается как теорема существования объекта, удовлетворяющего заданным соотношениям. Для этой теоремы строится конструктивное доказательство, т.е. такое, которое, утверждая существование объекта, формирует способ его вычисления. Из такого доказательства и извлекается текст программы.

Предлагаемый подход – это один из взглядов на то, как избавиться от семантического вакуума разрабатываемых сегодня систем, наполнив его предметной семантикой. Сделать библиотеки компонентов не просто пассивным хранилищем компонентов, а активными агентами, постоянно оптимизирующими входящие в них элементы. Предоставить экспертам возможность самостоятельно разрабатывать необходимое программное обеспечение с учетом их специфических потребностей.

Библиографический список

1. Czarnecki K. Generative Programming. Department of Computer Science and Automation, 1998. 429 p.

2. Kang K., Cohen S., Hess J., Nowak W. and Peterson S. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report, 1990.

3. Barker K., Clark P., Porter B. A Library of Generic Concepts for Composing Knowledge Bases. Department of Computer Sciences University of Texas at Austin, Austin. 8 p.

4. Schmidt D. Programming Language Semantics. Department of Computing and Information Sciences, Kansas State University, 1995. 20 p.

5. Mernik M., Heering J., Sloane A. When and how to develop domain-specific languages. Software Engineering Report SEN-E0517, 2005. 48 p.