Иркутский Государственный Технический Университет

Реферат

На тему: ***языки программирования***

Выполнил:
студент группы РДТв-94-1
Осипов И.Ф.

Руководитель:

Иркутск - 1998г.

Введение

Повышение производительности компьютеров и перемены в составе используемого ПО делают роль языков описания сценариев в создании приложении будущего все более и более важной. Эти языки отличаются от языков программирования системного уровня тем , что их основное назначение – связывать различные компоненты и приложения друг с другом , выполняя роль своего рода клея. В них находят применение бестиповые подходы к описанию данных, что позволяет вывести программирование на более высокии уровень и ускорить процесс разработки по сравнению с языками системного уровня.

За прошедшие 15 лет в методологии написания программ для компьютеров произошла радикальная перемена. Она состоит в том, что разработчики перешли от языков программирования системного уровня, таких как C и C++, к языкам описания сценариев, примерами которых могут служить Perl Tcl. Хотя в эту перемену оказалось вовлечено огромное количество людей , лишь немногие из них осознают , что в действительности происходит , и еще меньше найдется таких , кто бы смог объяснить причины.

Эти языки создавались для различных целей, что обусловило ряд фундаментальных различии между ним. Системные разрабатывались для построения структур данных и алгоритмов " с нуля", начиная от таких примитивных элементов , как слово памяти компьютера . В отличие от этого , языки описания сценариев создавались для связывания готовых программ . Их применение подразумевает наличие достаточного ассортимента мощных компонентов , которые требуется только объединить друг с другом . Языки системного уровня используют строгий контроль типов, что помогает разработчикам приложении справляться со сложными задачами; языки же описания сценариев не используют понятие типа , что упрощает установление связей между компонентами и ускоряет разработку прикладных систем.

Языки этих двух типов является взаимодополняющими , и большинство компьютерных платформ еще с середины 60-х годов оснащаются как теми , так и другими . В компонентных инфраструктурах они применяются , как правило , совместно компоненты создаются на языках программирования системного уровня , а для их связи между собой используются языки описания сценариев . Однако ряд современных тенденции , включая появление более быстрых машин и более совершенных языков описания сценариев , повышение значимости графического интерфейса пользователя и компонентных архитектур , а также рост популярности Internet, чрезвычайно расширили сферу применимости языков описания сценариев .Развитие этих тенденции продолжиться и в следующем десятилетии , вследствие чего все больше приложении будет создаваться целиком и полностью на языках описания сценариев , а роль языков программирования системного уровня сведется почти исключительно к созданию компонентов.

Языки программирования системного уровня

Чтобы осознать различие между языками описания сценариев и системными , полезно вспомнить историю развития последних. Впервые они появились в качестве альтернативы языкам ассемблера, позволяющим использовать в программе практически все особенности конкретной аппаратной подсистемы. Каждому утверждению такого языка соответствует ровно одна машинная команда, и программисту приходиться иметь дело с такими низко уровневыми деталями, как распределение регистров и последовательности вызова процедур. В результате написание и сопровождение крупных программ на языке ассемблера оказывается чрезвычайно сложным делом.

К концу 50-х годов начали появляться языки программирования более высокого уровня, такие как Lisp, Fortran, ALGOL. В них уже не было точного соответствия между языковыми конструкциями и машинными командами. Преобразование строк исходного кода в последовательности двоичных команд осуществлялось компилятором. Со временем их число пополнилось языками PL /1, Pascal, C, C++, Java. Все они менее эффективно используют аппаратуру по сравнению с языками ассемблера, но позволяет быстрее создавать приложения. В результате им удалось практически полностью вытеснить языки ассемблера при создании крупных приложении.

Языки программирования более высокого уровня

Языки программирования системного уровня отличаются от ассемблеров, во-первых, тем, что они являются более высокоуровневыми, и, во-вторых, используют более строгий контроль типов. Термин "высокоуровневый" означает следующее: многие детали обрабатываются автоматически, а программисту для создания своего приложения приходится писать меньшее количество строк. В частности:

- **Распределением регистров занимается компилятор, так что программисту не надо писать код, обеспечивающий перемещение данных между регистрами и памятью;**

- ***Последовательности вызова процедур генерируются автоматически; программисту нет необходимости описывать помещение аргументов функции в стек и их извлечение оттуда;***

- ***Для описания структур управления программист может использовать также ключевые слова, как if, while; последовательности машинных команд, соответствующие этим описаниям компилятор генерирует динамически.***

Типизация

Второе различие между языками ассемблера и языками программирования системного уровня состоит в типизации. Я использую этот термин для обозначения того, до какой степени значение информации бывает определено еще прежде, чем приходит время ее использования в программе. В сильно типизированных языках требуется, чтобы программист заранее декларировал способ использования каждого фрагмента информации, и затем уже языковые средства препятствуют применению ее каким-либо иным способом. В слабо же типизированных языках на способ обработки информации не налагается предварительных ограничении; интерпретация каждого элемента данных определяется только тем, как он фактически используется, без учета каких-либо предварительных объявлении.

Современные компьютеры устроены таким образом, что им не известно понятие типа. Каждое слово памяти может содержать значение любого типа; целое число, число с плавающей запятой, указатель или машинную команду. Интерпретация значения определяется способом его использования. Если указатель следующей машинной команды указывает в процессе исполнения машиннои команды на некоторое слово в памяти, то оно и рассматривается как команда; если адрес слова задан в параметрах команды целочисленного сложения, то его значение и обрабатывается как целое число; и т. д. Одно и то же слово памяти может использоваться в различных случаях разными способами.

В противоположность этому для современных языков программирования характерна строгая типизация. Каждая переменная в языке программирования системного уровня должна быть объявлена с указанием конкретного типа, такого как целое число или указатель на строку символов, и затем использоваться только соответствующими этому типу способами.

Данные и программный код разделены; создание нового кода по ходу исполнения программы затруднено, если вообще возможно. Переменные могут объединяться в структуры или объекты с четко определенной субструктурои и методами манипулирования своими компонентами. Объект одного типа не может быть использован в ситуации, где предписано применение объект другого типа.

Языки описания сценариев создавались для связывания готовых программ. Их применение подразумевает наличие достаточного ассортимента мощных компонентов, которые требуется только объединить друг с другом.

Типизация дает ряд преимуществ. Во-первых, крупные программы становятся благодаря ей более управляемыми. Четкость системы типов делает для программиста ясным, для чего предназначены те или иные данные; он легко может различать их между собой и соответственно использовать. Во-вторых, компиляторы используют информацию о типах для обнаружения некоторых видов ошибок, таких как попытка, использовать число с плавающей запятой в качестве указателя. В-третьих, типизация повышает производительность приложении, позволяя компиляторам генерировать более специализированный код. Например, если компилятору известно, что некоторая переменная всегда содержит целочисленные значения, он может генерировать для манипулирования ею целочисленные инструкции; если же тип переменой компилятору неизвестен, то приходиться вставлять дополнительные инструкции для проверки типа во время исполнения.

На рисунке 1 представлено распределение ряда языков программирования по мощности и степени строгости типизации.

Языки описания сценариев

Языки описания сценариев, такие как Perl, Python, Rexx, Tcl, Visual Basic и языки оболочек UNIX, предполагают стиль программирования, весьма отличныи от характерного для языков системного уровня. Они предназначаются не для написания приложении с “нуля”, а для комбинирования компонентов, набор которых создается заранее при помощи других языков. Например, Tcl, Visual Basic могут использоваться для построения пользовательских интерфеисов из имеющихся элементов управления, а языки описания сценариев для оболочек UNIX применяются для формирования “конвейеров” обработки потоков данных из набора стандартных фильтров. Языки описания сценариев часто применяются и для дополнения готовых компонентов новыми возможностями; однако эта деятельность редко охватывает создание сложных алгоритмов или структур данных, которые уже обычно бывают уже заложены в компоненты. Иногда языки описания сценариев даже называют связующими или языками системной интеграции.

Как правило, языки описания сценариев не типизированы

Для языков описания сценариев характерно отсутствие типизации, которая только усложнила бы задачу соединения компонентов. Все элементы в них выглядят и функционируют одинаково и являются полностью взаимозаменяемыми. Например, в Tcl или Visual Basic переменная может содержать в однои точке программы строку, а в другой – целое число. Код и данные также часто бывают взаимозаменяемы. Например, Tcl, Visual Basic переменная может содержать в одной точке программы строку, а в другой - целое число. Код и данные также часто бывают взаимозаменяемы, так что программа может генерировать другую программу — и сразу же запускать ее исполнение. Обычно языки описания сценариев используют переменные строковых типов, которые обеспечивают единообразный механизм представления для различных сущностей.

Отсутствие в языке деления переменных на типы упрощает соединение компонентов между собой. Нет априорных ограничении на то, каким образом может использоваться тот или иной элемент, а все компоненты значения представляются в едином формате. Таким образом, компонент или значение могут быть использованы в любой ситуации; будучи спроектированы для одних способов применения, они могут оказаться задействованы совершенно иными, о которых их создатель никогда не помышлял. Например, в UNIX – оболочках работа любой программы – фильтра включает чтение данных из входного потока и запись их в выходной поток. Любые две такие программы могут быть связаны путем назначения выходного потока одной в качестве входного потока другой. Следующая команда оболочки представляет систему из трех фильтров, подсчитывающую в выделенном фрагменте текста строки, содержащие слово “scipting”:

Select | grep scripting | wc

Программа *select* считывает текст, выделенный в данный момент на экране, и выводит его свои выходной поток; фильтр *grep* считывает входной поток и пропускает на выход строки, содержащие слово “*scripting”;* а программа *wc* подсчитывает число строк в своем потоке. Любой из подобных компонентов может найти применение во множестве различных ситуации, решая каждый раз иную общую задачу. Сильная типизация языков программирования системного уровня затрудняет повторное использование кода. Она поощряет программистов к созданию большого количества несовместимых друг с другом интерфейсов, каждый из которых требует применение объектов своего типа. Компилятор не позволяет объектам других типов взаимодействовать с этим интерфейсом, не смотря на то, что результат, мог бы оказаться и весьма полезным. Таким образом, чтобы использовать новый объект с существующем интерфейсом, программисту приходится писать “переходник”, преобразующий объект к типу, на который рассчитан интерфейс. А применение “переходника” требует, в свою очередь, перекомпиляции части или даже всего приложения целиком. Доминирующий в настоящее время способ распространения ПО в виде двоичных файлов делает этот подход невозможным.

Чтобы оценить преимущества бес типового языка программирования, рассмотрим следующий пример на

языке *Tcl:*

Button .b –text Hello! -font {Times 16} – comand {puts hello} .

Эта команда создает на экране новую кнопку с надписью на ней *Hello!* шрифтом *Times 16* пунктов, при нажатии, на которую выводится короткое сообщение *hello* . В одной строке здесь уместилось шесть элементов различных типов: название команды *(button),* название кнопки (*. b),* идентификаторы атрибутов *(-text, -font, -command)*, простые строки *(Hello! hello)*, спецификация шрифта *(Times 16),* состоящая из названия начертания *(Times)* и размера в пунктах (*16),* а также целый *Tcl*-сценарии (*puts hello).* Все элементы представляются единообразно – в виде строк. В данном примере атрибуты могли быть перечислены в произвольном порядке, а неупомянутым атрибутам (их насчитывается более 20) будут присвоены значения по умолчанию.

В случае реализации на *Java* тот же самый пример потребовал бы семи строк кода, составляющих два метода. Для *C++* с использованием библиотеки *Microsoft Foundation Classes (MFC)* масштабы увеличились примерно до 25 строк кода, образующих три процедуры. Один только выбор шрифта требует нескольких обращении к функциям*MFC*

Cfont *fontPtr=new Cront ();

fontPtr->CreteFont (16, 0, 0, 0, 700,

0, 0, 0, ANSI_CHARSET,

OUT_DEFAULT_PRECIS,

CLIP_DEFAULT_PRECIS,

DEFAULT_QUALITY,

DEFAULT_PITCH|

FF_DONTCARE,

“Times New Roman”);

buttonPtr->SetFont(fontPtr);

Можно было бы обоитись без значительнои части этого кода, если бы не строгая типизация. Чтобы задать шрифт для кнопки, необходимо обратиться к методу SetFont; однако он требует передачи в качестве аргумента указателя на объект CFont. Приходиться объявлять и инициализировать новый объект. Инициализацию объекта CFont выполняет его метод CreateFont, которыи имеет жесткии интерфеис, требующии задания 14 различных аргументов. В TCL существенные характеристики шрифта (начертание Times и кегль 16 пунктов) могут быть указаны непосредственно без каких-либо объявлении или преобразовании. Более того, TCL позволяет описать и поведение кнопки непосредственно в теле создающей ее команды, тогда как в C++ или Java для этого необходим отдельный метод.

Языки описания сценариев на подъеме

Языки описания сценариев существуют уже длительное время, однако, в последние годы в результате сочетания ряда факторов существенно повысилась их актуальность. Наиболее важный из этих факторов – направленность в сторону приложении, собираемых из готовых компонентов, В качестве трех иллюстрации его проявления можно назвать графические интерфейсы пользователя, Internet и компонентные инфраструктуры разработки.

Графические интерфейсы пользователя

Первые графические интерфейсы пользователя появились в начале 1980 г. и приобрели широкое распространение к концу десятилетия. Сегодня на описание этой части программы во многих проектах уходит более половины всех усилии разработчиков. GUI по своей природе является составной компонентной системой. Цель его создания состоит не в реализации новых функциональных возможностей, а в том, чтобы наладить связи между графическими элементами управления и функциями внутренних частей приложения.

Некоторые из систем снабжены очень удобными графическими средствами для построения экранов, которые скрывают сложности лежащего в основе языка, однако, как только возникает необходимость в написании дополнительного кода, например, чтобы расширить спектр вариантов поведения элементов интерфейса, у разработчика сразу возникают трудности. Все лучшие среды ускоренной разработки основаны на языках описания сценариев: Visual Basic, HyperCard, TCL/TK.

Internet

Развитие и рост популярности Internet также способствовали распространению языков описания сценариев. Сама сеть является не чем иным, как средством связи систем. Она не создает никаких новых данных и не занимается их обработкой; все, что она делает- обеспечивает легкий доступ к огромному множеству существующих объектов. Идеальным языком программирования для решения большинства связанных с сетью задач мог бы стать тот, который лучше организует совместную работу всех связанных компонентов, т. е. язык описания сценария. Так, для написания сеть-сценариев широко употребляется язык Perl, а среди разработчиков WEB-страниц популярен JavaScrint.

Компонентные инфраструктуры

Третий пример применения языков описания сценариев - компонентные инфраструктуры, такие как ActiveX, JavaBeans. Хотя языки программирования системного уровня с успехом используются для создания компонентов, задачи сборки из них приложении удобнее решаются при помощи сценариев. Без хорошего языка описания сценариев, предназначенного для манипулирования компонентами инфраструктуры, теряется значительная часть ее достоинств. Этим можно объяснить отчасти, почему компонентные инфраструктуры добились большей популярности в мире ПК, где существует такое удобное связующее средство, как Visual Basic, нежели на других платформах, таких как Unix/Cobra, компонентные инфраструктуры, для которых лишены языков описания сценариев.

Технология сценариев

Еще одна причина роста популярности языков описания сценариев – развитие их технологии. Такие современные представители этой категории, как TCL, Perl мало, чем напоминают своих далеких предшественников вроде JCL. Так, JCL не предусматривал даже простейших форм интерактивного взаимодействия, а ранние UNIX – оболочки не поддерживали процедур. Данная технология еще и сегодня остается относительно незрелой. Например, Visual Basic не является в полном смысле языком описания сценариев. Первоначально он был разработан в качестве упрощенного языка системного уровня, а затем – модифицирован так, чтобы его было удобнее применять к описанию сценариев. Таким образом, у будущих языков подобного рода есть большой простор для совершенствования.

Кроме того, технология сценариев много выиграла от повышения производительности компьютерного оборудования. Еще не так давно, чтобы добиться приемлемой скорости работы приложения любого уровня сложности, необходимо было обращаться к языкам системного уровня. В некоторых случаях даже их эффективность оказывалась недостаточной и программу приходилось писать на ассемблере. Современные машины работают в 100 – 500 раз быстрее компьютеров 80 – х годов, и их производительность продолжает удваиваться примерно каждые 18 месяцев. Сегодня целый ряд приложении может быть реализован на языках описания сценариев при тем не менее великолепной производительности. Например, TCL – сценарии позволяет манипулировать тысячами объектов при сохранении хорошего уровня интерактивности. По мере того как компьютеры будут становиться быстрее и быстрее, применение языков описания сценариев будет становиться привлекательным для реализации все более и более масштабных приложении.

Другие языки

Существует огромное количество атрибутов, помимо степени строгости контроля типов или уровня языка, и есть очень много интересных примеров, которые не могут быть однозначно отнесены к одной из двух рассмотренных нами категории. Например, семейство Lisp занимает некоторое промежуточное положение, обладая атрибутами языков описания сценариев и языков программирования системного уровня. В Lisp впервые были реализованы такие концепции, как интерпретация и динамический контроль типов, которые широко используются в современных языках описания сценариев, А также автоматическое управление хранением и интегрированные среды разработки, применяемые в языках обеих категории.

Языки описания сценариев основаны на несколько другом наборе компромиссов, чем языки системного уровня. В них скорость исполнения и строгость контроля типов ставятся в шкале приоритетов на более низкое место, но зато выше цениться производительность труда программиста и повторное использование. Это соотношение ценостеи оказывается все более оправданным по мере того, как компьютеры становятся быстродействующими и менее дорогими, чего нельзя сказать о программистах. Языки системного программирования хорошо подходят для создания компонентов, где основная сложность заключена в реализации алгоритмов и структур данных, тогда как языки описания сценариев лучше приспособлены для построения приложении из готовых компонентов, где сложность состоит в налаживании межкомпонентных связей. Задачи последнего рода получают все большее распространение, так что роль парадигмы сценариев будет возрастать и в будущем веке.