Труды IV Международной научно-технической конференции по телекоммуникациям (Телеком-99). Одесса, 1999. С. 116-119.

Контекстная технология программирования

В.С. Выхованец, к. т. н.

(Приднестровский государственный университет, Тирасполь)

В статье рассмотрена методологический базис, используемый при создании, сопровождении и развитие современных телекоммуникационных систем. В результате проведенного исследования выявлено, что перспективной является компонентная объектно-ориентированная технология программирования. Определены требования к компонентной технологии в целом и к используемому языку программирования. Предлагается в рамках компонентной технологии использование контекстных языковых средств, позволяющих снизить расходы на программирование. Приведены принципы построения контекстной технологии программирования на базе описываемого полуконтексного языка и вычислительной модели, представленной в виде трехстекового магазинного автомата.

In a paper is considered methodological base used at creation, support and development of modern telecommunication systems. The research is detected perspective component object-oriented process engineering of programming. The requirements to a component process engineering and to the used programming language are defined. It is offered within the framework of a component process engineering use of context-sensitive language tools permitting to lower expenditures on programming. The principles of a construction of a context-sensitive process engineering of programming based on the half-context language and computing model which represented as a three-push-down stack automaton.

Создание, сопровождение и развитие современных телекоммуникационных систем (ТС) базируется на методологии построения открытых систем [1], обладающих свойствами расширяемости (масштабируемости), мобильности (переносимости), интероперабельности и дружественности к пользователю [2]. С практической точки зрения успех проектирования ТС зависит от используемой методологии программирования - интегрального, сформулированного в виде принципов и методов подхода к основным стадиям проектирования программных средств.

Логика развития программной инженерии привела к объектно-ориентированной (ОО) парадигме. Успех применения ОО методологии зависит от используемой технологии программирования - системы способов и приемов разработки программных средств.

В результате исследования источников выявлено, что перспективной является компонентная объектно-ориентированная технология программирования, основанная на

следующих принципах: полная поддержка объектно-ориентированной методологии путем реализации децентрализации, контрактности, самодостаточности, классификации и бесшовности; независимость программного обеспечения от аппаратной и операционной платформ; динамическая встраиваемость модулей (компонент), созданных на различных платформах; расширяемость систем за счет миграции кода в распределенной компьютерной среде; динамическая генерация кода и трансляция методов мигрирующего кода по мере их вызова; анализ мигрирующей программы на предмет нарушения правил типизации и областей видимости; реализация средств контроля за уровнем безопасности для защиты от помех и несанкционированного вторжения; поддержка разработки крупных проектов.

Как следствие перечисленных принципов выявлены требования к объектно-ориентированному языку программирования. Последний должен обладать следующими свойствами: использование вычислительной модели, опирающаяся на взаимодействие "событие/обработчик"; отказ от директив препроцессора, указателей, оператора goto, неявного преобразования типов, адресной арифметики; реализация различных видов проверок во время исполнения (проверка на пустой указатель, контроль выхода индекса за границы массива, и т.д.); реализация одинарного наследования и строгой типизации; встроенный механизм обработки исключительных ситуаций и "сборки мусора"; наличие средств многопоточного программирования; нейтральный по отношению к архитектуре формат представления кода; выполнение независимой, раздельной и распределенной компиляции; разграничение описаний интерфейсной и реализационной части модуля; наличие выразительных средств для высокоуровневого (безопасного и переносимого) и низкоуровневого (опасными и непереносимыми) программирования; возможность задания мигрирующих программных компонентов; поддержка динамической генерации кода.

Узким местом современных технологий программирования остается объективная трудоемкость, интеллектуальная и технологическая сложность процесса программирования [3, 4].

В рамках компонентной технологии предлагается использование контекстных языковых средств, позволяющих снизить расходы на программирование. Реализована контекстная система программирования, удовлетворяющая требованиям к компонентной технологии. Наиболее близким аналогом разработанной системы является система программирования на базе языка FORTH [5].

Контекстный язык близок к естественным языкам и представлен в виде полуконтекстной формальной грамматики $G = \langle T, N, P, I \rangle$, где $T(N)$ - терминальный (нетерминальный)

алфавит, $T \cap N = \varnothing$; P - множество продукций вида: $\alpha A \to \beta \;\; \alpha, \beta \in \left(T \cup N\right)^*$, $A \in N$; I - аксиома.

Основными синтаксическими единицами языка является определение словаря <имя> и словарной статьи <слово>:

<словарь> ::= vocabulary <имя> <базовый> <статьи> end

<базовый> ::= <имя> <видимость>

<видимость> ::= public | private | protected

<статьи> ::= <статья> | <статья> <статьи>

<статья> ::= <слово> (<контекст> --- <контекст>) <определение> <видимость>

<контекст> ::= <имя> | <имя> <контекст>

<определение> ::= <лексема> | <лексема> <определение>

<лексема> ::= <имя> | <слово> | <слово> (<контекст>)

Словари образуют иерархическую структуру, соответствующую иерархии классов при ОО программировании. Поиск определения слова осуществляется по первому имени в текущем контексте, которое задает точку входа в дерево словарей. Выбор слова осуществляется по левому контексту. После компиляции слова, левый контекст заменяется на правый, что определяет семантику применения слова и изменяет контекст для следующего слова.

Контекст, задаваемый в определении слова обрабатывается компилятором и соответствует контексту, который должен быть после применения слова. Это позволяет конкретизировать выбор слова при полисемии, служит мощным средством для комментирования программы, а также обеспечивает со стороны компилятора контроль представлений программиста о выполняемой обработке данных.

Динамическое связывание реализуется путем поиска слова с заданным контекстом во время выполнения. Ниже приведен пример интерфейсной части словаря для работы с логическими данными:

```
vocabulary boolean object public
        boolean ( object --- boolean ) public
        ~ ( boolean --- boolean ) public
        & ( boolean boolean --- boolean ) public
        | ( boolean boolean --- boolean ) public
        ^ ( boolean boolean --- boolean ) public
        true ( --- boolean ) public
        false ( --- boolean ) public
```

```
end
```

В качестве вычислительного механизма использован магазинный автомат, имеющий 2 (3) стека: стек операндов, стек адресов возврата (стек контекста). Использование третьего стека позволяет реализовать динамическое связывание при произвольном контексте и произвести динамическую генерацию кода. При миграции кода подгрузка дерева прикладных словарей осуществляется в заданную точку базовой иерархии, начальная часть которой имеет вид:

```
vocabulary reference reference public
vocabulary object reference public
vocabulary boolean object public
vocabulary char object public
vocabulary memory object public
vocabulary byte memory public
vocabulary string memory public
vocabulary abstract memory public
vocabulary word abstract public
vocabulary vocabulary abstract public
```

Управляющие структуры языка (условный оператор, циклы do, while и for) определяются как слова словаря compile. Ниже приведен пример реализации слова read для словаря потокового ввода-вывода:

```
read ( size byte stream --- size )
        local stream this =             ( size byte )
        local byte ptr =                ( size )
        local size siz =                ( )
        local size len 0 =              ( )
        true                            ( boolean )
        begin
                len @ siz @ < &         ( boolean )
        while
                this cget dup           ( char char )
                ptr @ ! ptr ++ len ++   ( char )
                cr !=                   ( boolean )
        repeart
        len @ protected
```

Список используемой литературы

1. Зайцев С.С., Кравцунов М.И., Ротанов С.В. Сервис открытых информационно-вычислительных сетей: Справочник. М.: Радио и связь, 1990. [ITU-T (CCITT), Rec. X.200-X.219, 1988].

2. Филинов Е.Н. Выбор и разработка концептуальной модели среды открытых систем // Открытые системы. № 6. 1995. С. 71-77.

3. Lewis T. Software Architectures: Divine plan or digital Darwinism // Computer. № 8. 1996. P. 13-15.

4. Canceled Software Development Project costs Billions // Computer. № 8. 1995. P. 94.

5. Семенов Ю.А. Программирование на языке Форт. М.: Радио и связь, 1991.