УДК 519.681

КОНТЕКСТНАЯ МОДЕЛЬ ТЕХНОЛОГИЧЕСКОГО ПРОЦЕССА ПРЕДПРИЯТИЯ

В.Я. Иосенкин
Приднестровский государственный университет им. Т.Г. Шевченко
Молдова, 3300, Тирасполь, 25 Октября ул., 128
E-mail: slava@tiraet.com

В.С. Выхованец
Приднестровский государственный университет им. Т.Г. Шевченко
Молдова, 3300, Тирасполь, 25 Октября ул., 128
E-mail: slava@tiraet.com

Ключевые слова: контекстная модель, модель технологического процесса, моделирование процесса, контекст, контекстный язык, контекстная технология программирования, КТП, Эссе, объектно-ориентированный Форт, объектно-ориентированное программирование, контекстно-ориентированное программирование, компонентная технология, описание проблемной ситуации, предметная область, понятие, сущность, регулярное выражение, логическое программирование, управление, поддержка принятия решений.
Key words: contextual model, model of technological process, modeling of process, context, contextual language, contextual programming technology, CPT, Esse, the object-oriented Forth, object-oriented programming, context-oriented programming, component technology, the description of a problem situation, application domain, notion, essence, regular expression, logic programming, management, support of decision making.

Вводится понятие контекстной модели и модели технологического процесса в частности. Рассматривается построение модели с использованием контекстной технологии программирования и контекстного языка Esse. Приводится полная грамматика языка в форме Бэкуса-Науэра, структура сущностей. Рассматривается синтез контекстной модели системы управления железной дорогой, начиная с элементарных сущностей и заканчивая сложными прикладными понятиями. Делаются выводы о преимуществах новой технологии и соотношение ее с объектно-ориентированным подходом.
CONTEXTUAL ORIENTED MODEL OF ENTERPRISE TECHNOLOGICAL PROCESS / V.Y. Iosenkin (Dniester State University, 128 25th of October, Tiraspol 3300, Moldova, E-mail: slava@tiraet.com). The concept of contextual model and model of technological process in particular is entered. Construction of model with use of contextual programming technology and contextual language Esse is considered. The full grammar of language in Backus Naur form, essences structure is resulted. Synthesis of contextual model of a control system by railway, since elementary essences and finishing complex applied concepts is reviewed. Conclusions are drawn on advantages of new technology and its ratio with the object-oriented approach.

1. Введение

В последние годы интенсивно развивается подход к процессу информатизации предприятия, базирующийся на динамической модели системы поддержки принятия решений. Теперь идеальной или оптимальной структура управления может быть лишь для определенного момента развития мировой или локальной рыночной ситуации [1]. Начав проект внедрения, и описав потребности управленческой информационной системы, например, в январе, предприятие получит результаты лишь в июне. Затем, вероятно будет потрачено месяца два-три на утверждение руководством и внедрение, и к следующему январю предприятие получит идеальную и оптимальную модель управления… затем лишь, чтобы убедиться, что модель устарела как минимум на один год. Сегодня любое предприятие, стоящее у начала процесса внедрения информационной управленческой системы просто обязано думать о тех изменениях, которые произойдут с предприятием, страной и мировой экономической системой в то время, пока будет происходить внедрение, а также затем, после того, как внедрение закончено.

Одним из эффективных методов описания ситуативной информации является текст программы на контекстном языке [2]. Контекстный язык близок к естественным языкам и представлен в виде контекстной (или полуконтекстной) формальной грамматики G = <T,N,P,I>, где T(N) - терминальный (нетерминальный) алфавит, $T \cap N = \varnothing$; P - множество продукций вида: $\alpha A \rightarrow \beta$, где $\alpha, \beta \in (T \cup N)^*$, $A \in N$; I – аксиома [3-5]. Т.е. контекст - последовательность лексем, непосредственно предшествующая текущей лексеме.

Контекстной моделью будем называть описание некоторой производственной ситуации с использованием контекстных языковых средств [6], т.е. это текстовое описание системы, которое может дать ответ на наиболее важные с точки зрения понимания системы вопросы. Процесс моделирования какой-либо системы начинается с определения контекста, т.е. наиболее абстрактного уровня описания системы в целом. В контекст входит определение субъекта моделирования, цели и точки зрения на модель. Под субъектом понимается сама система, при этом необходимо точно установить, что входит в систему, а что лежит за ее пределами, другими словами, мы должны определить, что мы будем в дальнейшем рассматривать как компоненты системы, а что как внешнее воздействие. На определение субъекта системы будет существенно влиять позиция, с которой рассматривается система, и цель моделирования - вопросы, на которые построенная модель должна дать ответ. В отличие от других систем (например, Bpwin (IDEF0-модель) [7]) данная модель не требует обязательное наличие четко сформулированной цели, единственного субъекта моделирования и одной точки зрения.

Благодаря своим свойствам: компактность, легкая читабельность, высокая степень связанности элементов (изменение функциональности одной части модели сразу влечет за собой изменение функциональности других частей и всей модели в соответствии с установленными контекстными связями), высокая выразительная мощность, несложная сопровождаемость, надежность и переносимость, контекстная модель является удобным средством описания и управления технологическим процессом. Удачным объектом применения данной модели могут служить системы с высокой изменчивостью свих процессов, например, система управления железной дорогой, где имеется

множество железнодорожных путей, семафоров, поездов. Контекстная модель позволяет один раз описать все объекты взаимодействия с их свойствами программистом. И в дальнейшем требуется только лишь манипулирование ими в терминах данной предметной области, которое способен осуществить не программист, а квалифицированный в данной сфере пользователь. Т.е. его обязанности сводятся только к вводу маршрутов поездов и времени их отправления/прибытия. А диспетчеру (возможно, он и будет квалифицированным пользователем) остается лишь ввести некое событие, на которое система сама выдаст адекватное решение и исполнит его.

Главное преимущество данного метода заключается в высокой скорости разработки системы и чрезвычайно быстрой адаптации системы под изменившиеся условия, причем для последнего не требуется программист. Фактически процесс программирования и манипулирования сливаются в одно целое, где на первом этапе проектирования системы превалирует программирование, а на втором – манипулирование объектами.

2. Синтаксический анализ контекстной модели в языке программирования Esse

Представим обновленную грамматику контекстного объектно-ориентированного языка Esse (название языка происходит от англ. essence - сущность) в форме Бэкуса-Науэра:

```
program ::=         knowledge
                    knowledge program
                    '{' text '}'
                    '{' text '}' program
knowledge ::=       essence
                    essence '.' knowledge
essence ::=         name 'follows'
                    name 'growing' description
                    name ':' sentence
                    name declare
                    name declare ':' description
declare ::=         'is' visible notion
                    'as' visible notion
visible ::=         ''
                    'public'
                    'protected'
                    'private'
description ::=     sentence
                    sentence ';' description
sentence ::=        context definition
                    context ',' result definition
context ::=         notion local compile
                    notion local compile context
                    pattern local compile
                    pattern local compile context
                    literal compile
```

```
                literal compile context
local ::=       'as' name
compile ::=     '[' ']'
                '[' text ']'
result ::=      notion
                notion result
definition ::=  '{' '}'
                '{' text '}'
text ::=        block
                block '.' text
block ::=       narration
                narration ';' block
narration ::=   phrase
                phrase ',' narration
phrase ::=      term
                term phrase

literal ::=     ''' term '''
pattern ::=     '"' term '"'
notion ::=      name
name ::=        [a-z A-Z] [a-z A-Z 0-9_]*
term ::=        sign
                sign term
sign ::=        a-z A-Z 0-9 _ ( ) [ ] < > { } / \ ! ~ + - * % & | ^ ? $ // @ ` ' "

token ::=       blank prefix
                suffix blank
                term blank
blank ::=       ' ' | '\n' | '\r' | '\v'| '\f'
prefix ::=      '{' | '[' | '('
suffix ::=      '.'| ',' | ';' | ':' | '}' | ']' | ')'
```

Программа состоит из декларации знаний о предметной области (knowledge mode) и описания некоторой ситуации text (text mode), которая является исполняемой частью программы.

Знания о предметной области knowledge состоят из сущностей essence, разделенных точкой (как предложения в естественном языке).

Определяемые сущности essence образуют понятия notion с именами name находящиеся в отношении наследования. Понятие может быть объявлено без декларирования или описания (с помощью ключевого слова 'follows'). Это необходимо когда необходимо использовать еще неопределенное понятие (аналогично объявлению переменных в языках программирования). Данная конструкция резервирует понятие с именем name, описание которого будет далее по тексту программы.

Для того чтобы можно было использовать понятие в каких-либо операциях, его необходимо задекларировать (конструкция declare), т.е. определить его через другое понятие и задать его описание description. Конструкция declare определяет родство (наследование) понятий. Оно бывает двух типов: 'is' - понятие с именем name идентично (эквивалентно) понятию notion, 'as' - понятие

с именем name находится в отношении подобия (похоже, но имеет свои отличительные особенности) к понятию notion.

Учитывая, что язык Esse является объектно-ориентированным, он должен использовать и использует принципы наследования, инкапсуляции и полиморфизма. Каждое понятие находится в отношении эквивалентности или подобия с другим понятием, называемом родительским. Т.е. в результате формирования модели предметной области формируется дерево понятий, находящихся в родственных связях. Разумеется, на вершине этого дерева должно находиться понятие, которое не имеет родителя. Оно называется первичным понятием и является эквивалентным только самому себе. Первичные понятия определяются низкоуровневыми средствами, например, на языке ассемблера, и привязывают программу к конкретной вычислительной системе. Кроме того, каждое понятие имеет такой характерный для объектно-ориентированных языков атрибут, как видимость visible, соответственно он может принимать значение 'public', 'protected', 'private' или не иметь значения (что эквивалентно значению 'public'). Он позволяет более гибко обеспечивать или запрещать доступ к понятиям.

Каждое понятие определяется лексическими конструкциями sentence. В последних задается контекст их применения context, последовательность result понятий notion, производящих новый контекст context путем выполнения действий definition. В definition используются ранее описанные лексические конструкции и локальные значения понятий local из определяемой конструкции, посредством чего формируются значения нового контекста.

Контекст context представляет собой последовательность терминальных знаков term или понятий notion, их локальное значение можно сохранить в локальной переменной под псевдонимом name (с помощью ключевого слова 'as'), а также команд компилятору compile.Последовательность терминальных знаков при задании контекста может быть указана явно в виде литерала literal или с помощью шаблона pattern. Шаблон задается с помощью регулярных выражений, формат которых аналогичен имеющемуся в языке Perl, разработанный доктором Джоном Маддок [8]. Шаблон заключается в двойные, а литерал - одинарные кавычки.

Важно отметить тот факт, что в языке Esse программист может указать действия, которые необходимо выполнить, как на стадии исполнения (задаются конструкцией definition), так и компиляции (задаются конструкцией compile). Compile представляет собой конструкцию из терминальных знаков text, заключенных в квадратные скобки, а definition в фигурные. Text состоит из предложений block, разделенных точкой, которые в свою очередь состоят из повествований narration, разделенных точкой с запятой, которые состоят из фраз phrase, разделенных запятой, которые представляют собой последовательность терминальных знаков term, разделенных пробелом (в соответствии со структурой текстов на естественном языке).

Большую гибкость языку обеспечивает возможность ссылаться в конструкциях compile и definition на слова, используемые в context с помощью объявления локальных переменных (конструкция local). Но, кроме того, на слова context можно ссылаться с помощью специальных дополнительных конструкций %x и #x. %x – ссылка на понятие номер x в левой части context от compile или definition. Аналогично #x ссылается на соответствующий литерал. Нумерация понятий и лексем ведется с 0 слева направо.

Name является идентификатором, первым символом в котором может быть только буква, следующими буквы, цифра или символ подчеркивания.

Лексемы token, выделяемые лексическим анализатором, представляют собой последовательности знаков, разделенных пробельными знаками или знаками пунктуации, которые, в свою очередь, также являются лексемами.

3. Структура сущностей языка Esse

Понятия notion представляют собой структуру, содержащую помимо идентификатора понятия идентификатор родителя и вид родства с ним (импликация, 'is' или 'as'), область видимости, и предложения sentences, которые задают.

Предложение sentence представляет собой более сложную структуру. Идентификатором sentence является его порядковый номер, который присваивается автоматически по мере обработки кода программы и получения новых предложений, характеризующих данное понятие, и идентификатор понятия, к которому оно относиться. Порядковый номер предложения задает и его приоритет обработки. Прототип prototype предложения – список всех понятий, шаблонов и литералов, включенных в состав предложения, т.е. всех смысловых элементов предложения. Prototype используется интерпретатором во время процесса компиляции-интерпретации для служебных целей. Token в sentence – первая лексема (или шаблон) в предложении. В структуре sentence также хранятся локальных переменные предложения (local) и их количество (locals). Каждая локальная переменная имеет свое имя, являющееся ее идентификатором. Локальная переменная создается автоматически для каждой лексемы, встречающейся по тексту программы, ей присваивается имя в виде #x, а также для каждого понятия, имя в виде %x, где x - номер по порядку понятия или лексемы от начала предложения.

Количество лексем в предложении храниться в terms. Эти величины используются для навигации по стеку контекста [2].

Для хранения понятий и лексем, задающих контекст применения предложения, используется список backward, понятий и лексем предложения список forward (слова предложения), понятий, производящих новый контекст используется список produce. Коды операций (operational code), которые будут исполнены в результате применения предложения, хранятся в списке code.

Локальных переменные сохраняются в структуре, содержащей код операции, которую необходимо выполнить для локальной переменной, контекст, который должен присутствовать в стеке контекста, чтобы операция выполнилась и результат операции, т.е. список понятий, изменяющих контекст. Параметр param сохраняет относительное смещение локальной переменной в стеке.

В языке Esse существует единственная операция – операция @. Это операция, единственной целью которой является занесение значения, стоящего слева от нее, в стек для дальнейших манипуляций с ним. Все остальные операции не являются напрямую операциями языка Esse, а либо операции языка-родителя высокого уровня, либо низкоуровневые команды на ассемблере, либо функции операционной системы. Результатом работы интерпретатора является промежуточный код виртуальной машины, которая может быть реализована на любом языке и адаптирована под конкретную операционную систему и платформу, либо код на языке-родителе, тогда виртуальной машиной

будет являться компилятор или интерпретатор языка-родителя. Впрочем, в зависимости от реализации виртуальная машина может быть совмещена с интерпретатором, в случае если язык реализации контекстного языка программирования и язык, на котором выдается результирующий код, совпадают. Тогда, правда, ограничивается мобильность и переносимость языка [9].

4. Применение контекстной модели в технологическом процессе предприятия

Теперь попробуем описать некую предметную область, например систему управления движением поездов (СУДП), представленную в начале статьи. Очевидно, описание такой системы является достаточно сложной задачей, и оно включает в себя большое дерево достаточно многофункциональных и богатых внутренними и внешними связями сущностей, таких как, семафор, поезд, маршрут, машинист, диспетчер, авария, точка отправления, точка прибытия. время и т.д.

Для того, чтобы умело управлять производственным процессом необходимо четко описать все сущности и их взаимосвязи. Для этого вначале описываются элементарные сущности (по мере развития языка будут появляться различные библиотеки, описывающие различные предметные области), такие как булевы и арифметические операции, элементарные понятия языка на котором будет общаться пользователь с системой. Так, например, можно написать на языке Esse свой собственный язык для задания и управления какой-либо своей системой объектов, который будет наиболее удобен для манипулирования ею. Благодаря тому, что выразительная мощность языка равна выразительной мощности естественных языков, что определено его контекстной моделью, его можно использовать в более широком круге задач, нежели стандартные контекстно-свободные языки. Так на нем можно реализовать словари языка C++, Prolog, Lisp и языка управления поездами и переключаясь между словарями писать текст программы сразу на нескольких языках. Для примера приведем некоторую часть реализации языка C++ на языке Esse в плане простых арифметических операций (пример реализации булевых функций смотрите в [10]). Комментарии будем приводить после символов «//».

Прежде чем начинать описывать конкретные понятия и операции необходимо определить какие-то общие абстрактные понятия. Первичное понятие, как было указано ранее, определяется само через себя, назовем его literal.

```
literal is literal:
    literal '@' {};                    //Определяется первичная операция @-
                                       // операция: положить в стек.
    '".+"' as lit [lit @], literal {}. // Литерал – это любые символы в
                                       // кавычках, он кладется на вершину
                                       // стека для дальнейших манипуляций,
                                       // результат будет иметь тип literal.
name is literal:                       // Как видно, часть грамматики языка
                                       // описывается на самом этом языке.
    "[a-zA-Z][a-zA-Z0-9]*" [#0 @], name {}.  // Определяется, что есть name, -
```

```
                                        // слово, где первый символ буква, а
                                        // дальше буквы и цифры.
notion is name:                         // Определяется понятие
    '%' ['%' @], notion {};             // и операция %
    "%" [#0 @], notion {}.
value follows.                  // Вводится новое понятие value, которое
                                // будет определено далее по тексту.
```

Язык Esse использует стековый механизм передачи параметров, поэтому для него крайне важно иметь набор операций по работе со стеком.

```
local as notion:                        // Определение локальных переменных.
    'local' name [%0 @, 'local' @] {};  //  Определение слова 'local', как слова
                                        // для объявления типа local.
    local 'get' ['lget' @], value {};   // Определение функции get и put -
    value local 'put' ['lput' @] {}.    // извлечь и положить значение в стек.
global as local:                        // Определение глобальных
                                        // переменных. Глобальная переменная
                                        // – эта та же локальная переменная,
                                        // только с более широким спектром
                                        // использования.
    global 'get' ['gget' @], value {};
    value global 'put' ['gput' @] {};
    'global' name [%0 @, 'global' @] {};
    'global' name 'is' notion [%0 @, 'global' @, %0 %1 'notion'] {}.
value as global:                        //  Определение понятия value и
    value 'drop' ['drop' @];        // функций drop – удалить с вершины стека,
    value 'dup' ['dup' @], value value {};         // продублировать вершину
                                    // стека, результат - два value
    value "nop|inc|dec|not|neg" as code [code @], value {};
                                    // Определение унарных операций, один
                                    // аргумент, результат – один value.
value value "add|sub|and|or|xor|shl|shr|div|mul|mod" as code [code @], value {}.
// Определение бинарных операций, два аргумента на входе,
// один на выходе.
```

Как видно из приведенных примеров, одни и те же понятия вызывают разные действия (различные конструкции compile) и имеют разный результат (различные конструкции result) в зависимости от контекста, как левого, так и правого. Перегрузка операций и функций в языке C++ и ему подобных в определенной степени немного напоминают конструкцию compile. Однако, контекстное программирование значительно шире и предлагает более широкие возможности в плане задания конструкций, результат которых зависит от контекста. Так, при перегрузке операций в C++ программист ограничен количеством аргументов (например, для унарных - один, для бинарных операций - два) и порядком их следования (например, для бинарных – один слева, другой справа), нельзя вводить новые операции.

Теперь покажем, как реализуются арифметические операции с использованием контекстного программирования.

```
integer follows.                    // Вводиться новое понятие «целое», которое
                                    // будет определено далее по тексту.
numeric is local:                   // Задается базовый числовой тип.
    local as a 'is' 'integer' { a is 'numeric' }.

primary is value :                  // определяется первичное понятие – понятие
                                    // числа как такового и унарные операции
                                    // инкремент и декремент.
    numeric as a, value { numeric a get };
    numeric as a '++', primary { numeric a get; dup inc; numeric a put };
    numeric as a '--', primary { numeric a get; dup dec; numeric a put };
    '++' numeric as a, primary { numeric a get; inc dup; numeric a put };
    '--' numeric as a, primary { numeric a get; dec dup; numeric a put };
    "([\-+]?[0-9]+)", primary;      // Задается формат чисел с помощью
                                    // регулярного выражения.
    '(' integer ')', primary.       // Скобки тоже не будут лишними в
                                    // арифметических операциях.
```

При программировании создается впечатление, что программирование языка происходит на самом этом языке (потому что в конструкции definition можно использовать конструкции языка Esse, что придает дополнительную гибкость). Интересно то, что так оно и есть.

```
unary is primary :                  // Определение унарных операций.
    '-' primary, unary;
    '+' primary, unary;
    '~' primary, unary.
multiplication is unary :           // Определение бинарных операций.
    multiplication '*' unary, multiplication ;
    multiplication '/' unary, multiplication;
    multiplication '%' unary, multiplication.
addition is multiplication :
    addition '+' multiplication, addition;
    addition '-' multiplication, addition .
clearing is addition :
    clearing '&' addition, clearing.
exclusion is clearing :
    exclusion '^' clearing, exclusion.
setting is exclusion :
    setting '|' exclusion, setting.
shifting is setting :
    shifting '<<' setting, shifting;
    shifting '>>' setting, shifting.
comparing is shifting :
    shifting '>' shifting, boolean;
    shifting '<' shifting, boolean;
    shifting '==' shifting, boolean;
    shifting '!=' shifting, boolean;
    shifting '>=' shifting, boolean;
    shifting '<=' shifting, boolean;
```

```
    shifting '<>' shifting, boolean.
boolean follows.
ternary is comparing :
    boolean '?' ternary ':' ternary, ternary.          // тернарная операция ?:
integer is ternary :
    integer, boolean;
    numeric as a '=' ternary as b { b; numeric a put };
    numeric as a '*=' ternary as b { numeric a get, * b; numeric a put };
    // Аналогично определяются '/=', '%=',  '+=',  '-=', '&=', '|=', '^=', '<<=', '>>='.
```

В вышеописанном примере при описании контекста в бинарных операциях можно использовать, как понятие, определяемое в текущей конструкции, так и понятие-родителя. Это не имеет значения благодаря наследованию. В том числе их можно использовать вместе, что и делается в примере, для того, чтобы показать такую возможность.

Отметим также порядок задания операций, их описание задается в соответствии с их приоритетом – операции, имеющие более высокий приоритет, задаются первыми. Именно поэтому при задании контекста операций используются понятия, описывающие операцию, имеющую приоритет на один уровень выше, чем у той, которая в настоящий момент описывается.

После описания всех знаний, в данном случае операций и понятий, можно использовать запись числовых выражений, аналогично языку C++:

```
{
    integer x, integer y, integer z, integer xxx.
    x is 3, y is -4, z is 6;                    // можно использовать как «is»,
                                                // так и знак «=».
    xxx = ( x * x + y * y ) / -- z.             // xxx получает значение 5.
}
```

Описав все простые понятия и наполнив библиотеку базовыми сущностями, можно приступать к описанию более сложных - производственных технологических процессов предприятий. Особенно эффективно использовать данную технологию в условиях высокой динамики изменения алгоритмов работы предприятия и значительной специфики производства. Приведем упрощенный пример реализации части контекстной модели СУДП.

```
// Пусть будут определены следующие понятия.
global follows.                                 // Аналогично определенному выше.
object follows.                                 // Базовый объект,
                                                // родитель train.
time follows.                                   // Время – общее понятие.

// Объявим нижеследующие понятия.
situations is global:
    'situation' name {global %0 is situations},
    situations 'normal' {};
    situations 'abnormal' { /*сообщение диспетчеру */ };
    situations 'critical' { /*сообщение диспетчеру и начальникам близлежащих
станций */ };
```

situations 'accident' {/*сообщение диспетчеру и начальникам близлежащих станций, милицию и службу спасения */};

```
motion as global:                                   // Понятие «движение».
    motion, global;
    'forbid' ['false' @], motion {};
    'resolve' ['true' @], motion {};
    'undefined' ['NULL' @], motion {}.

semaphore as motion:                                // Понятие «семафор».
    semaphore, motion ['undefined' @];
    semaphore 'is' 'red', motion ['forbid' @] {};
    semaphore 'is' 'green', motion ['resolve' @] {}.

station as notion:                                  // Понятие «станция».
    station, notion;
    'station' notion, station [%0 @,'station' @] {}.

route as station:                                   // Цепочка из station.
    station '->' station , route;
    route '->' station, route.

ttime as time:                                  // Время.
    'departure' station 'at' time [%0 @, '%1 @] {}; // Время отправления.
    'arrival' station 'at' time [%0 @, %1 @] {}.    // Время прибытия.

ttimes as ttime:
    ttime, ttimes;
    ttime | ttime, ttimes.

train as object:                                // Понятие «поезд».
    'train' train, object;
    train 'stop' station [situation 'normal'];
    train 'stop' route [situation 'abnormal'];
    train 'goes' route [situation 'normal'];
    train 'goes' station [situation 'abnormal'].

location as route:
    train 'on' route, route [%0 @, %1 @].

times as ttimes:
    train '(' ttimes ')' {}.
```

В качестве исполняемой части (область определения переменных) может быть рассмотрен следующий пример.

```
{
    train t1; train t2; train t3;
    t1 on station1-> station2-> station3;
    t2 on station11-> station22-> station33;
```

t3 on station31-> station2-> station22->station32;

t1 (departure station1 at 07-00 | arrival station2 at 08-00 | departure station2 at 08-05 | arrival station3 at 09-00);

t2 (departure station11 at 07-00 | arrival station22 at 08-00 | departure station22 at 08-05 | arrival station33 at 09-00);

t3 (departure station31 at 07-00 | arrival station2 at 07-50 | departure station2 at 07-55 | arrival station22 at 08-15 | departure station22 at 08-20 | arrival station32 at 09-00);

}

После задания исходных данных управление поездами происходит автоматически без вмешательства человека. Единственное, что необходимо, чтобы от семафоров или диспетчеров поступала информация о текущем состоянии всех систем, а реакцию на них будет определять и выполнять сама программа.

5. Заключение

Представленный в статье язык является новым языком программирования, основная цель которого - приблизить процесс проектирования программных средств к тем парадигмам и сущностям, которые понятны конечному пользователю. Вторая задача, которую решает язык, - показать возможность и перспективность внедрения в процесс программирования контекстной технологии программирования (КТП). КТП не является чем выделенным и не зависимым – это технология, которая призвана дополнить и расширить грани объектно-ориентированной технологии и методологии. Она является ее логическим продолжением, ее надстройкой.

Подобно объектно-ориентированному программированию, она обеспечивает использование динамического связывания и наследования во время исполнения [11].

В отличие от объектно-ориентированного программирования:

1) Методы динамического связывания поддерживаются для всех типов данных, вернее сказать - для всех понятий;

2) Класс объекта (в КТП - понятие) может быть определен во время исполнения обычными средствами языка программирования.

3) Классы объектов и объекты могут быть заменены одним понятием «понятие», т.е. каждый объект будет являться одновременно и подклассом объектов, и от него всегда можно производить наследование. Но можно определить объекты и в чистом виде, как в объектно-ориентированных языках (см. определение train). В общем случае нет такого понятия, из которого нельзя было бы получить новых понятий-потомков.

4) Любой класс (понятие) определяется вместе со своим контекстом. Понятие, использующееся в неопределенном контексте, приводит к ошибке и не допускается [12]. Для того чтобы понятие использовалось независимо от контекста (чтобы использовать понятие аналогично тому, как они используются в контекстно-свободных языках) необходимо определить его без контекста.

Преимущества от внедрения новой технологии:

- повышение гибкости языка;
- повторное использование кода;

- динамическое связывание;
- применение методов динамического связывания к примитивным объектам;
- упрощение описания моделей различных техпроцессов;
- упрощение описания ситуации и адекватная немедленная реакция на него;
- облегчение диалога между системой управления и пользователем.

Построенная на основе данной технологии контекстная модель предоставляет широкие возможности по представлению структуры и механизмов различных технологических и управленческих процессов на предприятии в удобной форме восприятия (фактически в виде текстовой модели), но, кроме того, она в себе заключает широкие возможности по манипулированию ее объектами и управлению ими. Т.е. контекстная модель – это одновременно и модель технологического процесса и программа по управлению им.

Список литературы

1. Ястрежембский В.Р. Динамическая модель предприятия и Корпоративная Информационная Система нынешнего поколения // Управление и автоматизация. М. 1998.
2. Иосенкин В. Я., Выхованец В.С. Контекстное программирование в моделировании // Материалы международной научно-практической конференции «Моделирование. Теория, методы и средства». Новочеркасск, 2001. Часть 7. С. 49-51.
3. Выхованец В.С. Контекстная технология программирования // Труды IV Международной научно-технической конференции по телекоммуникациям (Телеком-99). Одесса, 1999. С. 116-119.
4. Выхованец В.С. Технология безопасного программирования // Тезисы докладов 8-й Международной конференции «Проблемы управления безопасностью сложных систем». М., 2000. Т. 2. С. 89-91.
5. Иосенкин В. Я., Выхованец В.С. Технология контекстного программирования в экономике и бизнесе // Управляющие и вычислительные системы. Новые технологии: Материалы межвузовской научно-технической конференции. – Вологда: ВоГТУ, 2001. – С. 180 – 181.
6. Иосенкин В.Я. Выхованец В.С. Применение технологии контекстного программирования для решения больших прикладных задач // Труды международной конференции «Параллельные вычисления и задачи управления» (PACO'2001). М.: Институт проблем управления им. В.А. Трапезникова РАН, 2001. С. (4)-121-139.
7. Марк Д.А, Гоуэн К.М. Методология структурного анализа и проектирования SADT. - Метатехнология, М. 1993.
8. Regex++, Index. http://www.boost.org/libs/regex/.
9. Иосенкин В.Я., Выхованец В.С. Технология контекстного программирования в телекоммуникационных системах // Труды второй международной научно-практической конференции «Современные информационные и электронные технологии». Одесса: Друк, 2001. С. 118-119.
10. Иосенкин В.Я. Качество и надежность проектирования программных средств // Надежность и качество. Труды международного симпозиума / Под ред. Н.К. Юркова. Пенза: Изд-во Пенз. гос. ун-та, 2002. С. 135-137.
11. Gassanenko M.L. Context-Oriented Programming. Proc. of the euroFORTH'98 conference, Marianske Lazne (Marienbad), Czech Republic.
12. Иосенкин В.Я. Контекстная интерпретация лексем // Материалы международной научно-практической конференции «Информационные технологии в науке и образовании». Шахты: Изд-во ЮРГУЭС, 2001. С. 73-75.