ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2016, № 1 (61)

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ ТА МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ

УДК 510.25:004.422.63.05

В. И. ШИНКАРЕНКО[1*], Г. В. ЗАБУЛА[2*]

[1*]Каф. «Компьютерные информационные технологии», Днепропетровский национальный университет железнодорожного транспорта имени академика В. Лазаряна, ул. Лазаряна, 2, Днепропетровск, Украина, 49010, тел. +38 (056) 373 15 35, эл. почта shinkrenko_vi@ua.fm, ORCID 0000-0001-8738-7225
[2*]Каф. «Компьютерные информационные технологии», Днепропетровский национальный университет железнодорожного транспорта имени академика В. Лазаряна, ул. Лазаряна, 2, Днепропетровск, Украина, 49010, тел. +38 (056) 373 15 35, эл. почта zabulus12@gmail.com, ORCID 0000-0002-8607-5729

КОНСТРУКТИВНАЯ МОДЕЛЬ АДАПТАЦИИ СТРУКТУР ДАННЫХ В ОПЕРАТИВНОЙ ПАМЯТИ: ЧАСТЬ I. КОНСТРУИРОВАНИЕ ТЕКСТОВ ПРОГРАММ

Цель. Стремительно растущие объемы обрабатываемых данных информационных систем существенно снижают временную эффективность алгоритмов их обработки. Одним из направлений решения данной проблемы является эффективное размещение данных в оперативной памяти (ОП). Необходимо разработать модели, позволяющие автоматизировано решать задачи рационального размещения данных в ОП. **Методика.** Для моделирования процессов адаптации структур данных (СД) в ОП применена методология математико-алгоритмического конструктивизма. Данный подход предусматривает разработку конструктивно-продукционных структур (КПС) с преобразованиями специализации, интерпретации, конкретизации и реализации. Разработка КПС предусматривает определение расширяемого носителя, сигнатуры отношений, операций и конструктивной аксиоматики. Наиболее сложной и существенной частью аксиоматики является множество формируемых правил подстановки, определяющих процесс вывода соответствующих конструкций. **Результаты.** Авторами разработана система КПС, состоящая из конструктора логической структуры данных, преобразователей логической структуры в программный интерфейс и имплементацию на языке программирования, конструкторов сценариев и процессов адаптации. Результатом реализации конструктора процесса адаптации являются генерации программного текста библиотеки классов, реализующей заданную логическую структуру данных с соответствующими операциями их обработки, и ее компиляция в бинарный код. **Научная новизна.** Впервые предложена конструктивная модель процессов разработки и адаптации структур данных к различным программно-аппаратным средам. При этом адаптируется размещение данных в ОП и алгоритмы их обработки. Применение конструктивизма в моделировании позволило в рамках единого подхода и применяемых средств связать модели данных и алгоритмы их обработки с критериями эффективности. Усовершенствована методика формирования системы КПС, механизмы, связи между взаимодополняющими друг друга КПС. Модификация конструктора и преобразователей позволяет коренным образом изменять и исследовать процесс адаптации. **Практическая значимость.** Разработанная модель позволяет автоматизировать процессы рационального размещения данных в ОП, что, в свою очередь, повышает временную эффективность программ со значительной долей обработки больших и очень больших объемов данных.

Ключевые слова: структура данных; конструктивно-продукционная структура; адаптация; конструктор; преобразователь

doi 10.15802/stp2016/61003

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2016, № 1 (61)

Введение

Существует несколько подходов формализации структур данных (СД). В большинстве из них модели данных не связаны с алгоритмами обработки. Такая связь прослеживается в работах [1, 2, 5], в которых заложены основы алгебраического аппарата, где в описание управляющих структур алгоритмов органично «встроены» данные. В [1] реализован комплексный подход на основе трехосновной алгебраической системы. При описании композиционных схем алгоритмы согласуются с управляющими структурами и СД. Описание СД в виде схем данных позволяет их детализовать, обеспечивая спецификацию данных на входе и выходе производных Д-операторов. СД могут быть реконфигурированы в соответствии с изменением алгоритма решаемой задачи.

Однако, в этом и других подходах отсутствует связь между логическим представлением данных и представлением данных в оперативной памяти в процессе выполнения обработки данных, что не позволяет решать задачи оптимизации и адаптации размещения СД в оперативной памяти (ОП) на основании критериев временной эффективности.

Проблемы формализации СД на логическом, физическом и промежуточных уровнях органично решаются средствами КПС [12].

Цель

Стремительно растущие объемы обрабатываемых данных информационных систем существенно снижают временную эффективность алгоритмов их обработки. Одним из направлений решения данной проблемы является эффективное размещение данных в оперативной памяти (ОП).

Инструментальные средства оптимизации программ не предусматривают изменений в размещении и соответствующей обработке данных в ОП. Некоторые приемы их эффективного проектирования [8-11, 14] требуют высокого уровня профессиональной подготовки специалистов и являются либо универсальными (не учитывающими особенностей программно-аппаратной среды их использования), либо узко специализированными.

Необходимы модели, позволяющие автоматизировано решать задачи рационального размещения данных в ОП с учетом программно-аппаратной среды их эксплуатации. При этом должны выполняться лишь допустимые преобразования [6] структур данных.

Методика

Обобщенной конструктивно-продукционной структурой (ОКПС) называется тройка [12]:

$$C_G =< M, \Sigma, \Lambda >,$$

где M – неоднородный носитель структуры; Σ – сигнатура, состоящая из множеств операций связывания, подстановки и вывода, операций над атрибутами и отношения подстановки; Λ – конструктивная аксиоматика. Аксиоматика Λ полностью представлена в [12].

Назначение конструкционно-продукционной структуры (КПС) состоит в формировании множеств конструкций с помощью операций связывания, подстановки, вывода и др. операций, задаваемых правилами аксиоматики.

В данной работе под конструкциями понимаются формируемые конструкции логических структур данных (ЛСД), тексты программ в описательной и исполнительной форме, сценарии обработки данных, конструктивные процессы адаптации.

Для формирования конструкций необходимо выполнять ряд уточняющих преобразований ОКПС [4, 12]:

− специализация определяет предметную область: семантическую природу носителя, конечное множество операций и их семантику, атрибутику операций, порядок их выполнения и ограничения на правила подстановки;

− интерпретация заключается в связывании операций сигнатуры с алгоритмами выполнения некоторой алгоритмической структуры [13]. При интерпретации выполняется связывание информационной модели способа построения конструкций и модели исполнителя;

− конкретизация КПС заключается в расширении аксиоматики множеством правил продукций, задании конкретных множеств нетерминальных и терминальных символов с их атрибутами и, при необходимости, значений атрибутов;

− реализация КПС заключается в формировании конструкции из элементов носителя КПС путем выполнения алгоритмов, связанных

doi 10.15802/stp2016/61003

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2016, № 1 (61)

с операциями сигнатуры. Реализация возможна только для предварительно специализированной, интерпретированной и конкретизированной КПС [12].

Согласно аксиоматике ОКПС формой ${}_{w}l$ с атрибутом w называется набор терминалов и нетерминалов, объединяемых операциями связывания. Конструкцией называется форма, содержащая только терминалы [12].

Правила подстановки имеют вид $\psi_r : \langle s_r, g_r \rangle \in \Psi$, где s_r – отношения подстановки; g_r – набор операций над атрибутами. Отношение подстановки – двуместное отношение с атрибутами ${}_{w_i}l_i \;\; {}_{w}\rightarrow \;\; {}_{w_j}l_j$ [12]. Для формы ${}_{w_l}l_l = {}_{w_0}\oplus({}_{w_1}l_1, {}_{w_2}l_2, ..., {}_{w_h}l_h, ..., {}_{w_k}l_k)$ и доступного отношения подстановки ${}_{w_h}l_h \;\; {}_{w_p}\rightarrow \;\; {}_{w_q}l_q$ такого, что ${}_{w_h}l_h \prec {}_{w_l}l_l$ (${}_{w_h}l_h$ является частью ${}_{w_l}l_l$), результатом трехместной операции подстановки ${}_{w_p}\Rightarrow({}_{w_h}l_h, {}_{w_q}l_q, {}_{w_l}l_l)$ будет форма ${}_{w_l^*}l_l^* = {}_{w_0}\oplus({}_{w_1}l_1, {}_{w_2}l_2, ..., {}_{w_q}l_q, ..., {}_{w_k}l_k)$ [12], где $\oplus$ – любая операция связывания из Σ.

Операция частичного вывода ${}_{maxn,\vec{q},m}l^* = (|\Rightarrow(\Psi, {}_{maxn,\vec{q},m}l))$ заключается в:

– выборе одного из доступных правил подстановки ${}_{\vec{d}}\psi_r : \langle s_r, g_r \rangle \in \Psi$, с отношениями подстановки s_r и выполнении на его основе операций подстановки, где $\vec{d}$ – вектор доступности правил. Доступность правила ${}_{\vec{d}}\psi_r$ определяется значением вектора доступности: если $d_r = 1$ правило доступно, если $d_r = 0$ – правило недоступно;

– выполнении операций над атрибутами g_r.

Порядок применения операции над атрибутами в процессе выполнения операции частичного вывода задается атрибутом τ_j, где $\tau_j \in I$, $I = \{\tau_0, \tau_1\}$, $I \subset M_{KAC}$, τ_0 – операция над атрибутом выполняется перед операцией подстановки, τ_1 – после операции подстановки.

Операция полного вывода (или просто вывода) заключается в последовательном выполнении операции частичного вывода, начиная с начального нетерминала и заканчивая конструкцией, удовлетворяющей условию окончания вывода.

Условием окончания вывода является отсутствие нетерминалов в форме.

Результаты

В первой части работы представлены вспомогательные конструкторы, позволяющие конструировать множество различных вариантов размещения данных в ОП и порождающие соответствующие им программы обработки данных, таких как добавление, поиск и т.д.

Конструктор логической структуры данных. Формальной структурой для проектирования СД на логическом уровне назовем КПС C_{LD}:

$$C =< M, \Sigma, \Lambda >_{S} \mapsto C_{LD} =< M_{LD}, \Sigma_{LD}, \Lambda_{LD} >, \quad (2)$$

где ${}_{S}\mapsto$ – операция специализации формальных структур (операция выполняется внешним исполнителем),

$$\Sigma_{LD} =< \Xi_{LD}, \Theta_{LD}, \Phi_{LD}, \{\rightarrow\} >, \; \Xi_{\mathrm{LD}} = \{\bullet, \odot, \div\},$$

$$\Lambda_{LD} = \Lambda_1 \bigcup \Lambda_2 \,.\, \Lambda_{LD} = \Lambda_1 \bigcup \Lambda_2 \,.$$

$$\Lambda_1 = \{M_{LD} \supset (T \bigcup N), T = \{{}_{\downarrow x_i, t_i, s_i} x_i\}, N = \{{}_{s_i}\alpha_i\}\},$$

где $\{x_i\}$ – множество простейших элементов данных с атрибутами: типом *type*, определяющим множество допустимых значений $\downarrow x_i$, множеством допустимых операций $Op = \{o_i\}$, семантикой ϕ_i; $\{\alpha_i\}$ – множеством нетерминалов.

Множество допустимых операций Op включает операции поиска, удаления, добавления и операции, построенные на их основе. Например, поиск по ключу, удаление нескольких элементов.

Частичная аксиоматика Λ_2 включает следующие аксиомы и инструктивные дополнения.

Операция связывания терминалов $x_i \bullet x_j$ обладает свойством симметричности:

$$\forall_{s_i} x_i \bullet {}_{s_i} x_j \;\&\; s_i \neq s_j \;\Rightarrow\; x_i \bullet x_j = x_j \bullet x_i \,.$$

Операция предназначена для связывания неоднородных элементов.

Операция связывания терминалов $x_i \odot x_j$ обладает свойством антисимметричности:

doi 10.15802/stp2016/61003

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2016, № 1 (61)

$$\forall_{s_i} x_i \odot_{s_i} x_j \;\&\; s_i = s_j \Rightarrow x_i \odot x_j \neq x_j \odot x_i .$$

Операция предназначена для связывания однородных элементов в примитивную конструкцию.

Значение трехместной операция условного выбора $\div(c,a,b)$ является a, если $c = true$ и b в противном случае.

Операции связывания имеют идентифицирующий атрибут $\rho_i \in P = \{\rho_1, \rho_2, ..., \rho_n\}$, где n – общее количество групп операций связывания в конкретизированной грамматике. Атрибут ρ_i применяется для идентификации операции связывания в различных частях ЛСД.

Для интерпретации C_{LD} построим модель исполнителя в виде базовой алгоритмической структуры (БАС) [13]:

$$C_{A,LD} =< M_{A,LD}, V_{A,LD}, \Sigma_{A,LD}, \Lambda_{A,LD} >,$$

где $M_{A,LD}$ – неоднородный носитель, $V_{A,LD}$ – множество образующих алгоритмов базовых (элементарных) для некоторого исполнителя, $\Sigma_{A,LD}$ – сигнатура и $\Lambda_{A,LD}$ – аксиоматика. Носитель $M_{A,LD} \supset T \cup N \cup \Omega(C_{A,LD}) \cup W$, где $\Omega(C_{A,LD})$ все сформированные алгоритмами алгоритмической структуры конструкции; W – множество допустимых значений атрибутов. Множество базовых $V_{A,LD} \supset \{A_1^0 |_{A_i,A_j}^{A_i \cdot A_j}$, $A_2^0 |_{Z_1,Z_2,A_i}^{:A_i}$, $A_3^0 |_{l_i,l_j}^{l_i \bullet l_j}$, $A_4^0 |_{l_i,l_j}^{l_i \odot l_j}\}$ и сконструированных $\{A_5 |_{f_h,f_q,f_i}^{f_j}$, $A_6 |_{f_i,\Psi}^{f_j}$, $A_7 |_{f_i,\Psi}^{f_j}\} \cup V_W \subset \Omega(C_{A,LD})$ алгоритмов:

– выполнения операции композиции алгоритмов $A_1^0 |_{A_i,A_j}^{A_i \cdot A_j}$ ($A|_X^Y$ – алгоритм над данными из входного множества X со значениями из множества Y, A^0 – образующий алгоритм), $A_i, A_j \in \Omega(C_{A,MS})$, $A_i \cdot A_j$ – последовательное выполнение алгоритма A_j после алгоритма A_i;

– условного выполнения алгоритма $A_2^0 |_{Z_1,Z_2,A_i}^{:A_i}$, который заключается в выполнении алгоритма A_i при условии $Z_1 \supseteq Z_2$;

– связывания неоднородных элементов $A_3^0 |_{l_i,l_j}^{l_i \bullet l_j}$, $l_i, l_j \in F^*$;

– связывания однородных элементов $A_4^0 |_{l_i,l_j}^{l_i \odot l_j}$, $l_i, l_j \in F^*$;

– выполнения операции подстановки $A_5 |_{l_h,l_q,f_i}^{f_j}$, $f_i, f_j \in F, l_h, l_q \in S$;

– выполнения операций частичного и полного вывода $A_6 |_{f_i,\Psi}^{f_j}$, $A_7 |_{f_i,\Psi}^{f_j}$ $f_i, f_j \in F$;

– выполнения операции условного выбора $A_8 |_{c,a,b}^{v}$;

– множество алгоритмов, реализующих операции над атрибутами V_W.

Интерпретация формальной структуры проектирования СД на логическом уровне:

$$< C_{LD} =< M_{LD}, \Sigma_{LD}, \Lambda_{LD} >,$$

$$C_{A,LD} =< M_{A,LD}, \Sigma_{A,LD}, \Lambda_{A,LD} >>$$

$$_I\mapsto \;\; {}_IC_{LD} =< M_{I,LD}, \Sigma_{I,LD}, \Lambda_{I,LD} >,$$

где $_I\mapsto$ – операция интерпретации;

$$\Lambda_{I,LD} = \Lambda_{LD} \bigcup \Lambda_3 ; \Lambda_3 = \{(A_3^0 |_{l_i,l_j}^{l_i \bullet l_j} \hookleftarrow \bullet);$$

$$(A_4^0 |_{l_i,l_j}^{l_i \odot l_j} \hookleftarrow \odot);\; (A_5 |_{l_h,l_q,f_i}^{f_j} \hookleftarrow \Rightarrow);\; (A_6 |_{f_i,\Psi}^{f_j} \hookleftarrow |\Rightarrow);$$

$$(A_7 |_{f_i,\Psi}^{f_j} \hookleftarrow \|\Rightarrow);\; (A_8 |_{c,f,t}^{v} \hookleftarrow \div);\; \forall \circ_i \in \Phi : \exists$$

$$(A_i |_{X_i}^{Y_i} \hookleftarrow "\circ_i "),\;\; A_i |_{X_i}^{Y_i} \in \Omega(C_{A,LD})\}.$$

Рассмотрим одну из конкретизаций $_IC_{LD}$ на примере логической структуры BMP файла [7]

$$_IC_{LD} =< M_{I,LD}, \Sigma_{I,LD}, \Lambda_{I,LD} >$$

$$_K\mapsto \;\; C_{L_{BMP}} =< M_{L_{BMP}}, \Sigma_{L_{BMP}}, \Lambda_{L_{BMP}} >,$$

где $_K\mapsto$ – операция конкретизации;

$$M_{L_{BMP}} \supset T_{LD} \cup N_{LD} \cup \mathbb{N};$$

$$\Sigma_{L_{BMP}} = \Sigma_{I,LD} \cup \Sigma_{BMP};$$

$$\Sigma_{BMP} = \{"\leq","\geq","<",">",":=","+"\};$$

doi 10.15802/stp2016/61003

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2016, № 1 (61)

$\Lambda_{L_{BMP}} = \Lambda_{I,LD} \cup \Lambda_4 \cup \Lambda_5$. $\Lambda_4 =< T_{LD} = \{a,\ b,\ c,\ e,\ f,\ o,\ q,\ z,\ x,\ v\},\ N_{LD} = \{\alpha,\beta,\delta,\eta,\varphi,\theta\} >$.

Частичная аксиоматика Λ_5 включает в себя следующие правила подстановки.

Отношение s_1 определяет общую структуру файла

$$s_1 =< {}_{\phi}\alpha \ \ {}_{d_1}\!\rightarrow\ {}_{\phi}\beta\ {}_{\rho_1}\!\bullet\ {}_{\phi}\delta >;$$

$${}_{\tau_0 \lrcorner \tau} g_1 :=< d_1 := \text{true} > .$$

Сокращенный информационный заголовок BMP-файла представлен s_2. Заголовок состоит из высоты, ширины, количества бит на пиксель

$$s_2 =< {}_{\phi}\beta \ \ {}_{d_2}\!\rightarrow\ {}_{\text{type},\phi}a\ {}_{\rho_2}\!\bullet\ {}_{\text{type},\phi}b\ {}_{\rho_2}\!\bullet\ {}_{\text{type},\phi}c >;$$

$${}_{\tau_0 \lrcorner \tau} g_2 :=< d_2 := \text{true} > .$$

Следующее отношение определяет данные изображения в виде последовательности цветов в RGB-модели.

$$s_3 =< {}_{\phi_1}\delta \ \ {}_{d_3}\!\rightarrow\ {}_{type,\phi}e\ {}_{\rho_5}\!\odot\ {}_{\phi}\eta >;$$

$${}_{\tau_0 \lrcorner \tau} g_3 :=< d_3 := \div(z \lrcorner a > 256, \text{true}, \text{false}) > .$$

Далее представляется изображение в виде модели Index Color

$$s_4 =< {}_{\phi_2}\delta \ \ {}_{d_4}\!\rightarrow\ {}_{\phi}\varphi\ {}_{\rho_3}\!\bullet\ {}_{\phi}\theta >;$$

$${}_{\tau_0 \lrcorner \tau} g_4 :=< d_4 := \div(z \lrcorner a \le 256, \text{true}, \text{false}) > .$$

Отношение s_5 определяет структуру палитры. Палитра представлена в виде последовательности цветов в RGB-модели

$$s_5 =< {}_{\phi}\varphi \ \ {}_{d_5}\!\rightarrow\ {}_{\text{type},\phi}f\ {}_{\rho_6}\!\odot\ {}_{\phi}\eta >;$$

$${}_{\tau_0 \lrcorner \tau} g_6 :=< d_6 := \text{true} > .$$

Данные изображения в виде индексов, указывающих на элементы палитры, определены в следующем отношении

$$s_6 =< {}_{\phi}\theta \ \ {}_{d_6}\!\rightarrow\ {}_{\text{type},\phi}p\ \ {}_{\rho_7}\!\odot\ {}_{\text{type},\phi}q >;$$

$${}_{\tau_0 \lrcorner \tau} g_6 :=< d_6 := \text{true} > .$$

Отношение s_7 определяет цвет в модели RGB

$$s_7 =< {}_{\phi}\eta \ \ {}_{d_7}\!\rightarrow\ {}_{\text{type},\phi}z\ \ {}_{\rho_3}\!\bullet\ {}_{\text{type},\phi}x\ \ {}_{\rho_3}\!\bullet\ {}_{\text{type},\phi}v >;$$

$${}_{\tau_0 \lrcorner \tau} g_7 :=< d_7 := \text{true} > .$$

Значения атрибутов семантики приведены в табл. 1. Идентификаторы, указанные в табл. 1, используются для именования полей СД в преобразователях, приведенных далее.

Таблица 1

Table 1

Атрибут	Значение	Идентификатор
$\phi \lrcorner \alpha$	BMP файл без файлового заголовка	BMPFile
$\phi \lrcorner \beta$	Информационный заголовок BMP	InfoHeader
$\phi_1 \lrcorner \delta$	Данные изображения в модели RGB	ImageDataRGB
$\phi \lrcorner a$	Количество битов на пиксель	BitsPerPixel
$\phi \lrcorner b$	Ширина изображения	Width
$\phi \lrcorner c$	Высота изображения	Height
$\phi \lrcorner f$	Массив палитры цвета RGB	Palette
$\phi \lrcorner e$	Цвета RGB	RGBMatrix
$\phi_2 \lrcorner \delta$	Данные изображения в модели индексированного цвета	ImageDataIndex
$\phi \lrcorner \delta$	Данные изображения	ImageData
$\phi \lrcorner p$	Матрица индексов цвета пикселей	IndexMatrix
$\phi \lrcorner q$	Индекс цвета пикселей	Index
$\phi \lrcorner \eta$	Цвет RGB	RGB
$\phi \lrcorner z$	Компонента R	R
$\phi \lrcorner v$	Компонента B	B

В результате реализации могут быть сформированы конструкции ЛСД, соответствующие BMP файлу, с моделью цвета:

– Index Color

$$({}_{\text{type},\phi}a\ {}_{\rho_2}\!\bullet\ {}_{\text{type},\phi}b\ {}_{\rho_2}\!\bullet\ {}_{\text{type},\phi}c)\ {}_{\rho_1}\!\bullet((\ {}_{\text{type},\phi}f\ {}_{\rho_6}\!\odot$$

doi 10.15802/stp2016/61003

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2016, № 1 (61)

$({}_{\text{type},\phi} z \ {}_{\rho_3}\bullet\ {}_{\text{type},\phi} x \ {}_{\rho_3}\bullet\ {}_{\text{type},\phi} v))\ {}_{\rho_4}\bullet\ {}_{\text{type},\phi} p \ {}_{\rho_7}\odot\ {}_{\text{type},\phi} q)$;

– True Color

$$({}_{type,\phi} a \ {}_{\rho_2}\bullet\ {}_{type,\phi} b \ {}_{\rho_2}\bullet\ {}_{type,\phi} c)\ {}_{\rho_1}\bullet$$

$$({}_{type,\phi} e \ {}_{\rho_5}\odot({}_{type,\phi} z \ {}_{\rho_3}\bullet\ {}_{type,\phi} x \ {}_{\rho_3}\bullet\ {}_{type,\phi} v)).$$

Преобразователь логической структуры данных в программный интерфейс на языке программирования. После проектирования на абстрактном и логическом уровнях следует этап проектирования СД на уровне представления (языке программирования) [7, 15].

Специализированной формальной структурой для преобразования логической СД в представление на языке программирования, C_{PLI} будет:

$$C =< M, \Sigma, \Lambda > {}_S\mapsto\ C_{PLI} = $$

$$=< M_{S,PLI}, \Sigma_{S,PLI}, \Lambda_{S.PLI} >,$$

где

$$\Sigma_{S,PLI} =< \Xi_{PLI}, \Theta_{PLI}, \Phi_{PLI}, \{\rightarrow\} >,$$

$$\Xi_{PLI} = \{\bullet, \odot\} \cup \Xi_1,$$

$$\Phi_{PLI} = \varnothing,$$

$$\Lambda_{S,PLI} = \Lambda \cup\ \Lambda_6 \cup \Lambda_7,$$

$$\Lambda_6 = \{M_{PLI} \supset\ (T_{S,PLI} \bigcup N_{S,PLI}),$$

$$T_{S,PLI} = \{C_{\#}\},\quad N_{S,PLI} = \{!a_i!\},$$

где $\{C_{\#}\}$ – ключевые слова, знаки операций и разделители языка программирования C#, Ξ_1 – включает единственную операцию конкатенации лексем языка программирования, знак операции везде опускается, $!a_i!$ – нетерминалы правил подстановки.

Частичная аксиоматика Λ_7 определяет состав правил подстановки: $\psi_i : \langle s_i \rangle \in \Psi$, $s_i =< s_{i,1}, s_{i,2} >$, где $s_{i,1}$ – правило подстановки логической структуры данных, $s_{i,2}$ – соответствующее правило подстановки для формирования текста программы.

Интерпретацию C_{PLI} выполним, используя ранее определенную алгоритмическую структуру

$$C_{A,LD} =< M_{A,LD}, \Sigma_{A,LD}, \Lambda_{A,LD} > :$$

$$< C_{S,PLI} =< M_{S,PLI}, \Sigma_{S,PLI}, \Lambda_{S,PLI} >,$$

$$C_{A,LD} =< M_{A,LD}, \Sigma_{A,LD}, \Lambda_{A,LD} >>$$

$${}_I\mapsto\ {}_I C_{PLI} =< M_{I,PLI}, \Sigma_{I,PLI}, \Lambda_{I,PLI} >$$

где $\Lambda_{I,PLI} = \Lambda_{PLI} \cup \Lambda_{A,LD}$;

Конкретизируем структуру следующими правилами подстановки

$${}_I C_{PLI} =< M_{I,\text{PLI}}, \Sigma_{I,\text{PLI}}, \Lambda_{I,\text{PLI}} >$$

$${}_K\mapsto\ C_{PLI} =< M_{PLI}, \Sigma_{PLI}, \Lambda_{PLI} >,$$

где

$$M_{PLI} \supset T_{PLI} \cup N_{PLI};$$

$$\Sigma_{PLI} = \Sigma_{I,PLI};$$

$$\Lambda_{PLI} = \Lambda_{I,PLI} \cup \Lambda_8 \cup \Lambda_9.$$

$\Lambda_8 =<\ T_{PLI} =\ T_{LD} \cup\ T_{S,PLI} \cup \{$ IBMPFile, IInfoHeader, IImageData, BitsPerPixel, Width, Height, IMatrixRGB, IRGB, IImageDataIndex, IPalette, IIndexMatrix, IMatrixIndex, R, G, B$\}$, $N_{PLI} =$

$$N_{LD} \cup \{\ !\text{BMPFILEDECL}!,$$

!INFOHEADERDECL!, !IMAGEDATADECL!, !INFOHEADERTYPE!, !IMAGEDATATYPE!, !INFOHEADEREDECL!, !IMAGEDATATYPE!, !IMAGEDATADECL!, !RGBDECL!, !RGBTYPE! $\}$ >.

Частичная аксиоматика Λ_9 включает следующие правила подстановки.

Интерфейс файла BMP, состоит из информационного заголовка и данных. Оба поля имеют соответствующий тип, декларация которого представлено следующими правилами.

$$s_1 =< s_{1,1}, s_{1,2} >;$$

$$s_{1,1} =<\ {}_{\phi_\alpha}\alpha\ \ {}_{p\nu_0}\rightarrow\ {}_{\phi_\beta}\beta_{\rho_0}\bullet\ {}_{\phi_\delta}\delta >;$$

$$s_{1,2} =<$$

!BMPFILEDECL! ->
!INFOHEADERDECL! !IMAGEDATADECL!
interface IBMPFile {

doi 10.15802/stp2016/61003

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2016, № 1 (61)

!INFOHEADERTYPE! InfoHeader {get;set;}
!IMAGEDATATYPE! ImageData {get;set;}} > ,

Интерфейс информационного заголовка состоит из трех полей.

$$s_2 =< s_{2,1}, s_{2,2} > ;$$

$$s_{2,1} =< {}_{\phi_\beta}\beta \rightarrow {}_{t_a\phi_a m_a} a_{\rho_1} \bullet {}_{t_b\phi_b m_b} b_{\rho_1} \bullet {}_{t_c\phi_c m_c} c > ;$$

$$s_{2,2} =<$$

!INFOHEADERTYPE! -> IInfoHeader
!INFOHEADEREDECL ->
interface IInfoHeader {
int BitsPerPixel {get;set;}
int Width {get;set;}
int Height {get;set;} } > ,

Структура данных изображения в виде модели True Color. Используется встроенный в .NET Framework интерфейс IList.

$$s_3 =< s_{3,1}, s_{3,2} > ;$$

$$s_{3,1} =< {}_{\phi_{1,\delta}}\delta \rightarrow {}_{\phi_e v_e} e \odot \eta > ;$$

$$s_{3,2} =<$$

!IMAGEDATATYPE! -> IMatixRGB
!IMAGEDATADECL! ->
!RGBDECL!
interface IMatrixRGB : IList< IList< !RGBTYPE! > > { } > .

Структура данных изображения в виде модели Index Color состоит из полей палитры и матрицы индексов

$$s_4 =< s_{4,1}, s_{4,2} > ;$$

$$s_{4,1} =< {}_{\phi_{2,\delta}}\delta \rightarrow {}_{\phi_\varphi}\varphi_{\rho_2} \bullet {}_{\phi_\theta}\theta > ;$$

$$s_{4,2} =<$$

!IMAGEDATATYPE! -> IImageDataIndex
!IMAGEDATADECL! ->
!INDEXMATRIXDECL!
!PALETTEDECL!
interface IImageDataIndex {
!PALETTETYPE! Palette {get;set;}
!INDEXMATRIXTYPE! IndexMatrix {get;set;}
} > .

Интерфейс палитры на основе IList интерфейсы

$$s_6 =< s_{6,1}, s_{6,2} > ;$$

$$s_{6,1} =< {}_{\phi_\varphi}\varphi \rightarrow {}_{v_f\phi_f} f \odot \eta > ;$$

$$s_{6,2} =<$$

!PALETTETYPE! -> IPalette
!PALETTEDECL! ->
!RGBDECL!
interface IPalette : IList<!RGBTYPE!> { } > .

Интерфейс матрицы индексированных цветов.

$$s_7 =< s_{7,1}, s_{7,2} > ;$$

$$s_{7,1} =< {}_{\phi_\theta}\theta \rightarrow {}_{v_p\phi_p} p \odot {}_{v_q\phi_q} q > ;$$

$$s_{7,2} =<$$

!INDEXMATRIXTYPE! -> MatrixIndex
!INDEXTYPE! -> byte
!INDEXMATRIXDECL! ->
interface IMatrixIndex : IList< IList< !INDEXTYPE! > > { } > ,

$$s_8 =< s_{8,1}, s_{8,2} > ;$$

$$s_{8,1} =< {}_{\phi_\pi}\eta \rightarrow {}_{\phi_z v_z} z \bullet {}_{\phi_x v_x} x \bullet {}_{\phi_v v_v} v > ;$$

$$s_{8,2} =<$$

!RGBTYPE! -> IRGB
!RGBDECL! ->
inteface IRGB {
int R{get;set;}
int G{get;set;}
int B{get;set;} } > .

В результате реализации преобразователя формируется интерфейс классов обработки структуры данных на языке программирования C#. Одна из таких реализаций приведена ниже.

```
interface IInfoHeader {
int BitsPerPixel { get; set; }
int Width { get; set; }
int Height { get; set; } }
interface IRGB {
int R { get; set; }
int G { get; set; }
int B { get; set; } }
interface IMatrixRGB : IList<IList<IRGB>> { }
interface IBMPFile {
IInfoHeader InfoHeader { get; set; }
IMatrixRGB ImageData {get;set;} }
```

doi 10.15802/stp2016/61003

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)
Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2016, № 1 (61)

Преобразователь абстрактной логической структуры данных в представление имплементации на языке программирования. Специализированной формальной структурой для преобразования ЛСД в представление языка программирования назовем КПС C_{PL}:

$$C =< M, \Sigma, \Lambda > \ {}_S\mapsto \ C_{S,PL} =< M_{S,PL}, \Sigma_{S,PL}, \Lambda_{S,PL} >,$$

где ${}_S\mapsto$ –операция специализации формальных структур (операция выполняется внешним исполнителем),

$$\Sigma_{S,PL} =< \Xi_{PL}, \Theta_{PL}, \Phi_{PL}, \{\rightarrow\} >,$$

$$\Xi_{\mathrm{PL}} = \{\bullet, \odot\},$$

$$\Lambda_{S,PL} = \Lambda_{10} \cup \Lambda_{11} \cup \Lambda_{12}.$$

$$\Lambda_{10} = \{\mathrm{M}_{PL} \supset (T_{S,PL} \cup N_{S,PL}),$$

$$T_{S,PL} = \{C_{\#}\},$$

$$N_{S,PL} = \{!a_i!\},\}.$$

Частичная аксиоматика Λ_{11} содержит следующие конструктивные дополнения.

Правила подстановки преобразователя имеют вид: $\psi_r : \langle s_r, g_r \rangle \in \Psi$, $s_r =< s_{l,1}, s_{r,2} >$. Где $s_{l,1}$ – правило вывода логической структуры данных, $s_{r,2}$ – соответствующее правило вывода текста программы, g_r – атрибуты преобразования правил вывода ЛСД.

Частичная аксиоматика Λ_{12} содержит следующие конструктивные дополнения.

Конструкция преобразователя имеет вид пары: $< f_{C_{PL}}, \vec{R} >$, где $f_{C_{PL}}$ – реализация преобразователя – текст программы имплементации операций обработки СД; $\vec{R} =< r_1, r_2, ..., r_k >$ r_i – количество допустимых программных шаблонов для i-ой составляющей СД; k –количество примитивных конструкций в ЛСД.

Нетерминалы содержат атрибут $\vec{Q} =< q_1, q_2, ..., q_k >$, где q_i – номер конкретного программного шаблона для генерации текста программ, $0 \le q_i \le r_i$.

Интерпретацию C_{PL} выполним, используя ранее определенную алгоритмическую структуру $C_{A,LD} =< M_{A,LD}, \Sigma_{A,LD}, \Lambda_{A,LD} >$.

Преобразователь логической структуры данных представляет собой модель программного интерфейса. Он является интерпретацией формальной структуры C_{PL} алгоритмической структурой $C_{A,PL}$:

$$< C_{S,PL} =< M_{S,PL}, \Sigma_{S,PL}, \Lambda_{S,PL} >,$$

$$C_{A,LD} =< M_{A,LD}, \Sigma_{A,LD}, \Lambda_{A,LD} >>$$

$${}_I\mapsto \ {}_I C_{SC} =< M_{I,SC}, \Sigma_{I,SC}, \Lambda_{I,SC} >$$

Программным шаблоном будем называть код на языке C#, который реализует примитивную конструкцию тем или иным способом (например, используя физическое представление данных в виде массива, списка, и т.п.).

Конкретизируем структуру следующими правилами подстановки

$${}_I C_{PL} =< M_{I,PL}, \Sigma_{I,PL}, \Lambda_{I,PL} >$$

$${}_K\mapsto \ C_{PL} =< M_{PL}, \Sigma_{PL}, \Lambda_{PL} >,$$

где

$$M_{PL} \supset N_{PL} \cup T_{PL};$$

$$\Sigma_{PL} = \Sigma_{I,PL};$$

$$\Lambda_{PLI} = \Lambda_{I,PL} \cup \Lambda_{13} \cup \Lambda_{14}.$$

$\Lambda_{13} = \{ \ T_{PL} = T_{PLI} \cup \{$ BMPFile, InfoHeader, ImageData, MatrixRGB, RGB, ImageDataIndex, Palette, IndexMatrix, MatrixIndex$\}$,

$$N_{PL} = N_{PLI},$$

$$\vec{R} =< 1,2,1,1,0,1,1,2 > \}.$$

Частичная аксиоматика Λ_{14} включается в себя следующие правила подстановки.

$$s_1 =< s_{1,1}, s_{1,2} >;$$

$$s_{1,1} =< {}_{\vec{Q},\phi_\alpha}\alpha \ \rightarrow \ {}_{\vec{Q},\phi_\beta}\beta_{\rho_0} \bullet \ {}_{\vec{Q},\phi_\delta}\delta >;$$

$$s_{1,2} =<$$

!BMPFILEDECL! ->
!INFOHEADERDECL! !IMAGEDATADECL!
class BMPFile {
!INFOHEADERTYPE! InfoHeader {get;set;}
!IMAGEDATATYPE! ImageData {get;set;} }
BMPFile root = new BMPFile(); > ,

doi 10.15802/stp2016/61003

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2016, № 1 (61)

$$g_1 =< d_1 = \div(q_1 = 0, \text{true}, \text{false}) > ;$$

$$s_2 =< s_{2,1}, s_{2,2} > ;$$

$$s_{2,1} =< {}_{\vec{Q},s_\beta}\beta \rightarrow {}_{t_a\phi_a m_a} a_{\rho_1} \bullet {}_{t_b\phi_b m_b} b_{\rho_1} \bullet {}_{t_c\phi_c m_c} c > ;$$

$$s_{2,2} =<$$

!INFOHEADERTYPE! -> InfoHeader
!INFOHEADEREDECL! ->
class InfoHeader {
int BitsPerPixel {get;set;}
int Width {get;set;}
int Height {get;set;}} > ,

$$g_2 =< d_2 = \div(q_2 = 0, \text{true}, \text{false}) >$$

$$s_3 =< s_{2,1}, s_{3,2} > ;$$

$$s_{3,2} =<$$

!INFOHEADEREDECL!->
class InfoHeader {
LinkedList<int> internal = new LinkedList <int>();
LinkedListNode<int> GetNthNode(int index) {
var first = internal.First;
int i = 0;
while(i < index) {
first = first.Next;
++I;
}
return first;
}
public InfoHeader() {
internal.AddLast(0); // BPP
internal.AddLast (0); // W
internal.AddLast(0); // H
}
int BitsPerPixel {
get { return internal.Skip(0).First(); }
set { this.GetNthNode(0).Value = value; }
}
int Width
get { return internal.Skip(1).First(); }
set { this.GetNthNode(1).Value = value; }
}
int Height {
get { return internal.Skip(2).First(); }
set { this.GetNthNode(2).Value = value; } } } > ,

$$g_3 =< d_3 = \div(q_2 = 1, \text{true}, \text{false}) > ;$$

$$s_4 =< s_{4,1}, s_{4,2} > ;$$

$$s_{4,1} =< {}_{\vec{Q},\phi_{1,\delta}}\delta \rightarrow {}_{\phi_e v_e} e \odot {}_{\vec{Q},\phi_\eta}\eta > ;$$

$$s_{4,2} =<$$

!IMAGEDATATYPE! -> MatixRGB
!IMAGEDATADECL! ->
!RGBDECL!
class MatrixRGB :
List<List<!RGBTYPE!>> { } > ,

$$g_4 =< d_4 = \div(q_3 = 0, \text{true}, \text{false}) > ;$$

$$s_5 =< s_{5,1}, s_{5,2} > ;$$

$$s_{5,1} =< {}_{\vec{Q},\phi_{2,\delta}}\delta \rightarrow {}_{\vec{Q},\phi_\varphi}\varphi_{\rho_2} \bullet {}_{\vec{Q},\phi_\theta}\theta > ;$$

$$s_{5,2} =<$$

!IMAGEDATATYPE! -> ImageDataIndex
!IMAGEDATADECL! ->
!INDEXMATRIXDECL!
!PALETTEDECL!
class ImageDataIndex {
!PALETTETYPE! Palette {get;set;}
!INDEXMATRIXTYPE! IndexMatrix {get;set;} } > ,

$$g_5 =< d_5 = \div(q_4 = 0, \text{true}, \text{false}) > ;$$

$$s_6 =< s_{6,1}, s_{6,2} > ;$$

$$s_{6,1} =< {}_{\vec{Q},\phi_\varphi}\varphi \rightarrow {}_{v_f\phi_f} f \odot {}_{\vec{Q},\phi_\eta}\eta > ;$$

$$s_{6,2} =<$$

!PALETTETYPE! -> Palette
!PALETTEDECL! ->
!RGBDECL!
class Palette : Vector<*!RGBTYPE!*> { } > ,

$$g_6 =< d_6 = \div(q_6 = 0, \text{true}, \text{false}) > ;$$

$$s_7 =< s_{7,1}, s_{7,2} > ;$$

$$s_{7,1} =< {}_{\vec{Q},\phi_\theta}\theta \rightarrow {}_{v_p\phi_p} p \odot {}_{v_q\phi_q} q > ;$$

$$s_{7,2} =<$$

!INDEXMATRIXTYPE! -> MatrixIndex
!INDEXTYPE! -> int
!INDEXMATRIXDECL! ->
class MatrixIndex : Matrix<!INDEXTYPE!> { } > ,

doi 10.15802/stp2016/61003

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2016, № 1 (61)

$$g_7 =< d_7 = \div(q_7 = 0, \text{true}, \text{false}) > ;$$

$$s_8 =< s_{8,1}, s_{8,2} > ;$$

$$s_{8,1} =< {}_{\vec{Q},\phi_\pi}\eta \rightarrow {}_{v_z\phi_z} z \bullet {}_{v_x\phi_x} x \bullet {}_{v_v\phi_v} v > ;$$

$$s_{8,2} =<$$

!RGBTYPE! -> RGB
!RGBDECL! ->
class RGB {
int R{get;set;}
int G{get;set;}
int B{get;set;} }

$$g_8 =< d_8 = \div(q_8 = 0, \text{true}, \text{false}) > ;$$

$$s_9 =< s_{8,1}, s_{9,2} > ;$$

$$s_{9,2} =<$$

!LISTIMPL_DECLARE_INT_LIST! -> List<int> internalList = new List<int>();
!LISTIMPL_FIELD_BEG! -> public int
!LISTIMPL_FIELD_MI1! -> {
get{return internalList [
!LISTIMPL_FIELD_MI2! ->];}
set { internalList [
!LISTIMPL_FIELD_END! ->] = value; }}
!RGBDECL! ->
class RGB
{
!LISTIMPL_DECLARE_INT_LIST!
public RGB() { }
!LISTIMPL_FIELD_BEG! R
!LISTIMPL_FIELD_MI1! 0
!LISTIMPL_FIELD_MI2! 0
!LISTIMPL_FIELD_END!
!LISTIMPL_FIELD_BEG! G
!LISTIMPL_FIELD_MI1! 1
!LISTIMPL_FIELD_MI2! 1
!LISTIMPL_FIELD_END!
!LISTIMPL_FIELD_BEG! B
!LISTIMPL_FIELD_MI1! 2
!LISTIMPL_FIELD_MI2! 2
!LISTIMPL_FIELD_END!

$$g_9 =< d_9 = \div(q_8 = 1, \text{true}, \text{false}) > .$$

Пусть внешним исполнителем задан вектор $\vec{Q} =< 1,2,1,1,0,1,1,2 >$, тогда при реализации будет сформирована следующая имплементация программного кода на языке C#:

```
class InfoHeader {
LinkedList<int> internalList = new LinkedList<int>();
LinkedListNode<int> GetNthNode(int index) {
var first = internalList.First;
int i = 0;
while (i < index) {
first = first.Next;
++i;
}
return first;
}
public InfoHeader() {
internalList.AddLast(0); // BPP
internalList.AddLast(0); // W
internalList.AddLast(0); // H }
int BitsPerPixel {
get { return internalList.Skip(0).First(); }
set { this.GetNthNode(0).Value = value; } }
int Width {
get { return internalList.Skip(1).First(); }
set { this.GetNthNode(1).Value = value; } }
}
int Height {
get { return internalList.Skip(2).First(); }
set { this.GetNthNode(2).Value = value; } } }
class RGB {
List<int> internalList = new List<int>();
public RGB() {
internalList.Capacity = 3;
}
int R {
get { return internalList[0]; }
set { internalList[0] = value; } }
int G {
get { return internalList[1]; }
set { internalList[1] = value; } }
int B {
get { return internalList[1]; }
set { internalList[1] = value; } } }
class MatrixRGB : List<List<RGB>> { }
class BMPFile {
InfoHeader InfoHeader { get; set; }
MatrixRGB ImageData {get;set;} }
```

Выводы

Представленные конструктор логической структуры данных, преобразователи логической структуры в программный интерфейс и имплементация на языке программирования – являются неотъемлемой частью системы КПС. Основная часть системы в виде конструкторов

doi 10.15802/stp2016/61003

сценариев и процессов адаптации представлена во второй части статьи.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Акуловский, В. Г. Алгебра для описания данных в композиционных схемах алгоритмов / В. Г. Акуловский // Проблеми програмування. – 2012 – № 2-3. – С. 234–240.
2. Акуловский, В. Г. Основы алгебры алгоритмов, базирующейся на данных / В. Г. Акуловский // Проблеми програмування. – 2010. – № 2-3. – С. 89–96.
3. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. – Санкт-Петербург : ООО «И. Д. Вильямс», 2011. – 1296 с.
4. Босов, А. А. Структурная сложность систем / А. А. Босов, В. М. Ильман, Н. В. Халипова // Вісн. Дніпропетр. нац. ун-ту залізн. трансп ім. акад. В. Лазаряна. – Дніпропетровськ, 2012. – Вип. 40. – С. 173–179.
5. Дорошенко, А. Е. Алгебра алгоритмов с данными и прогнозирование вычислительного процесса / А. Е. Дорошенко, В. Г. Акуловский // Проблеми програмування. – 2011. – № 3. – С. 3–10.
6. Дрожжин, В. В. Преобразование структур данных в поле структур данных / В. В. Дрожжин, А. М. Володин // Изв. Пензен. гос. пед. ун-та им. В. Г. Белинского. – Пенза, 2011. – № 26. – С. 380–385.
7. Шинкаренко, В. И. Конструкционно-продукционная модель структур данных на логическом уровне / В. И. Шинкаренко, В. М. Ильман, Г. В. Забула // Проблеми програмування. – 2014. – № 2-3. – С. 10–16.
8. Array Based HV/VH Tree: an Effective Data Structure for Layout Representation / J. Ren, W. Pan, Y. Zheng [et al.] // J. of Zhejiang University-SCIENCE C (Computers & Electronics). – 2012. – Vol. 13. – Iss. 3. – P. 232–237. doi: 10.1631/jzus.c1100193.
9. Attali, D. Efficient Data Structure for Representing and Simplifying Simplicial Complexes in High Dimensions / D. Attali, A. Lieutier, D. Salinas // Intern. J. of Computational Geometry & Applications. – 2012. – Vol. 22. – Iss. 4. – P. 279–303. doi: 10.1142/S0218195912600060.
10. Bentley, J. L. Writing Efficient Programs / J. L. Bentley. – New Jersey : Prentice-Hall in Englewood Cliffs, 1982. – 170 p.
11. Drepper, U. What Every Programmer Should Know About Memory / U. Drepper. – Raleigh : RedHat, Inc., 2007. – 114 p.
12. Shynkarenko, V. I. Constructive-Synthesizing Structures and Their Grammatical Interpretations. I. Generalized Formal Constructive-Synthesizing Structure / V. I. Shynkarenko, V. M. Ilman // Cybernetics and Systems Analysis. – 2014. – Vol. 50. – Iss. 5. – P. 655–662. doi: 10.1007/s10559-014-9655-z.
13. Shynkarenko, V. I. Structural Models of Algorithms in Problems of Applied Programming. I. Formal Algorithmic Structures / V. I. Shynkarenko, V. M. Ilman, V. V. Skalozub // Cybernetics and Systems Analysis. – 2009. – Vol. 45. – Iss. 3. – P. 329–339. doi: 10.1007/s10559-009-9118-0.
14. Weiss, M. A. Data Structures and Algorithm Analysis in C++ / M. A. Weiss. – New Jersey : Pearson Education Inc., Addison-Wesley, 2014. – 656 p.
15. Ziegler, C. A. Programming System methodologies / C. A. Ziegler. – New Jersey : Prentice-Hall, Englewood Cliffs, 1983. – 260 p.

В. І. ШИНКАРЕНКО[1*], Г. В. ЗАБУЛА[2*]

[1*]Каф. «Компьютерні інформаційні технології», Дніпропетровський національный університет залізничного транспорту імені академіка В. Лазаряна, вул. Лазаряна, 2, Дніпропетровськ, Україна, 49010, тел. +38 (056) 373 15 35, ел. пошта shinkrenko_vi@ua.fm, ORCID 0000-0001-8738-7225

[2*]Каф. «Компьютерні інформаційні технології», Дніпропетровський національный університет залізничного транспорту імені академіка В. Лазаряна, вул. Лазаряна, 2, Дніпропетровськ, Україна, 49010, тел. +38 (056) 373 15 35, ел. пошта zabulus12@gmail.com, ORCID 0000-0002-8607-5729

КОНСТРУКТИВНА МОДЕЛЬ АДАПТАЦІЇ СТРУКТУР ДАНИХ В ОПЕРАТИВНІЙ ПАМ'ЯТІ: ЧАСТИНА І. КОНСТРУЮВАННЯ ТЕКСТІВ ПРОГРАМ

Мета. Стрімко зростаючі обсяги оброблюваних даних інформаційних систем істотно знижують часову ефективність алгоритмів їх обробки. Одним із напрямків вирішення даної проблеми є ефективне розміщення даних в оперативній пам'яті (ОП). Необхідно розробити моделі, що дозволяють автоматизовано вирішувати

doi 10.15802/stp2016/61003

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2016, № 1 (61)

завдання раціонального розміщення даних в ОП. **Методика.** Для моделювання процесів адаптації структур даних (СД) в ОП застосована методологія математико-алгоритмічного конструктивізму. Даний підхід передбачає розробку конструктивно-продукційних структур (КПС) із перетвореннями спеціалізації, інтерпретації, конкретизації та реалізації. Розробка КПС передбачає визначення розширюваного носія, сигнатури відносин та операцій і конструктивної аксіоматики. Найбільш складною та істотною частиною аксіоматики є безліч формованих правил підстановки, що визначають процес виводу відповідних конструкцій. **Результати.** Авторами розроблено систему КПС, що складається з конструктора логічної структури даних, перетворювачів логічної структури в програмний інтерфейс й імплементацію на мові програмування, конструкторів сценаріїв та процесів адаптації. Результатом реалізації конструктора процесу адаптації є генерації програмного тексту бібліотеки класів, що реалізує задану логічну структуру даних із відповідними операціями їх обробки, та її компіляція в бінарний код. **Наукова новизна.** Вперше запропонована конструктивна модель процесів розробки та адаптації структур даних до різних програмно-апаратних середовищ. При цьому адаптується розміщення даних в ОП та алгоритми їх обробки. Застосування конструктивізму в моделюванні дозволило в рамках єдиного підходу та застосовуваних засобів зв'язати моделі даних і алгоритми їх обробки з критеріями ефективності. Удосконалено методику формування системи КПС, механізми, зв'язки між взаємодоповнюючими один одного КПС. Модифікація конструктора та перетворювачів дозволяє докорінно змінювати і досліджувати процес адаптації. **Практична значимість.** Розроблена модель дозволяє автоматизувати процеси раціонального розміщення даних в ОП, що, у свою чергу, підвищує часову ефективність програм зі значною часткою обробки великих і дуже великих обсягів даних.

Ключові слова: структура даних; конструктивно-продукційна структура; адаптація; конструктор; перетворювач

V. I. SHYNKARENKO[1*], H. V. ZABULA[2*]

[1*]Dep. «Computer and Information Technologies», Dnipropetrovsk National University of Railway Transport named after Academician V. Lazaryan, Lazaryana St., 2, Dnipropetrovsk, Ukraine, 49010, tel. +38 (056) 373 15 35, e-mail shinkrenko_vi@ua.fm, ORCID 0000-0001-8738-7225
[2*]Dep. «Computer and Information Technologies», Dnipropetrovsk National University of Railway Transport named after Academician V. Lazaryan, Lazaryana St., 2, Dnipropetrovsk, Ukraine, 49010, tel. +38 (056) 373 15 35, e-mail zabulus12@gmail.com, ORCID 0000-0002-8607-5729

CONSTRUCTIVE MODEL OF DATA STRUCTURES ADAPTATION IN RAM: PART I. PROGRAM TEXT CONSTRUCTING

Purpose. Rapidly growing volumes of information systems data being manipulated significantly reduce the temporary algorithms effeciency of their processing . Effective data layout in RAM is one of the directions of solving this problem. It is necessary to develop the model to solve problems of efficient automated data layout in RAM. **Methodology.** For processes simulation of data structures (DS) adaptation in RAM, the methodology of mathematical and algorithmic constructivism was applied. This approach involves the development of constructive and productive structures (CPS) with transformations of specialization, interpretation, specification and implementation. CPS development provides definition of expandable vector, signature of relations, transactions and constructive axioms. The most complex and essential part of the set of axioms is generated substitution rules that determine the output process of respective structures. **Findings.** CPS system was developed by the authors, consisting of the logical structure constructor of data, converters of logical structure in to a software interface and implementation in a programming language, constructors of scenarios and adaptation processes. The result of the adaptation process constructor is software text generations of the class library that implements the specified logical data structure with appropriate processing operations and its compilation in binary code. **Originality.** Structural model of development processes and data structures adaptation to different software and hardware environments was first proposed. It adapts date layout in the RAM and data processing algorithms. Application of constructivism in simulation allowed within a single approach and applied tools linking the data models and algorithms of their processing with performance criteria. Formation methodology of CPS system, mechanisms, and links between complementary CPS were improved. Modification of the constructor and converters allows changing and exploring the process of adaptation. **Practical value.** The developed model allows automating the data layout in RAM, which in turn increases the time efficiency of programs with significant processing of large and very large volumes of data.

Keywords: data structure; constructive and productive structure; adaptation; constructor; converter

doi 10.15802/stp2016/61003

REFERENCES

1. Akulovskiy V.G. Algebra dlya opisaniya dannykh v kompozitsionnykh skhemakh algoritmov [Algebra for describing the data in the compositional schemes of algorithms]. *Problemy prohramuvannia – Problems in Programming*, 2012, no. 2-3, pp. 234-240.
2. Akulovskiy V.G. Osnovy algebry algoritmov, baziruyushcheysya na dannykh [Basic algebra algorithms based on data]. *Problemy prohramuvannia – Problems in Programming*, 2010, no. 2-3, pp. 89-96.
3. Kormen T., Leyzerson Ch., Rivest R., Shtayn K. *Algoritmy: postroeniye i analiz* [Algorithms: construction and analysis]. Saint-Petersburg, OOO «I. D. Vilyams» Publ., 2011. 1296 p.
4. Bosov A.A., Ilman V.M., Khalipova N.V. Strukturnaya slozhnost sistem [Structural complexity of systems]. *Visnyk Dnipropetrovskoho natsionalnoho universytetu zaliznychnoho transportu imeni akademika V. Lazariana* [Bulletin of Dnipropetrovsk National University of Railway Transport named after Academician V. Lazaryan], 2012, issue 40, pp. 173-179.
5. Doroshenko A.Ye., Akulovskiy V.G. Algebra algoritmov s dannymi i prognozirovaniye vychislitelnogo protsessa [Algebra of algorithms with data and prediction computational process]. *Problemy prohramuvannia – Problems in Programming*, 2011, no. 3, pp. 3-10.
6. Drozhzhin V.V., Volodin A.M. Preobrazovaniye struktur dannykh v pole struktur dannykh [Convert data structures in the field of data structures]. *Izvestiya penzenskogo gosudarstvennogo pedagogicheskogo Universiteta im. V.G. Belinskogo* [News of Penza State Pedagogical University named after V. G. Belinsky], 2011, no. 26, pp. 380-385.
7. Shinkarenko V.I., Ilman V.M., Zabula G.V. Konstruktsionno-produktsionnaya model struktur dannykh na logicheskom urovne [Construction-production model of the data structures at the logical level]. *Problemy prohramuvannia – Problems in Programming*, 2014, no. 2-3, pp. 10-16.
8. Ren J., Pan W., Zheng Y., Shi Z., Yan X. Array Based HV/VH Tree: an Effective Da-ta Structure for Layout Representation. *Journal of Zhejiang University-SCIENCE C (Computers & Electronics)*, 2012, vol. 13, issue 3, pp. 232-237. doi: 10.1631/jzus.c1100193.
9. Attali D., Lieutier A., Salinas D. Efficient Data Structure for Repre-senting and Simplifying Simplicial Complexes in High Dimensions. *Intern. Journal of Computational Geometry & Applications*, 2012, vol. 22, issue 4, pp. 279-303. doi: 10.1142/S0218195912600060.
10. Bentley J.L. Writing Efficient Programs. New Jersey, Prentice-Hall in Englewood Cliffs Publ., 1982. 170 p.
11. Drepper U. What Every Programmer Should Know About Memory. Raleigh, RedHat, Inc. Publ., 2007. 114 p.
12. Shynkarenko V.I., Ilman V.M. Constructive-Synthesizing Structures and Their Grammatical Interpretations. I. Generalized Formal Constructive-Synthesizing Structure. *Cybernetics and Systems Analysis*, 2014, vol. 50, issue 5, pp. 655-662. doi: 10.1007/s10559-014-9655-z.
13. Shynkarenko V.I., Ilman V.M., Skalozub V.V. Structural Models of Algorithms in Problems of Applied Programming. I. Formal Algorithmic Structures. *Cybernetics and Systems Analysis*, 2009, vol. 45, issue 3, pp. 329-339. doi: 10.1007/s10559-009-9118-0.
14. Weiss M.A. Data Structures and Algorithm Analysis in C++. New Jersey, Pearson Education Inc., Addison-Wesley Publ., 2014. 656 p.
15. Ziegler C.A. Programming System methodologies. New Jersey, Prentice-Hall, Englewood Cliffs Publ., 1983. 260 p.

Статья рекомендована к публикации д.т.н., проф. В. В. Скалозубом (Украина); д.физ.-мат.н., проф. В. Е. Билозёровым (Украина)

Поступила в редколлегию: 14.12.2015
Принята к печати: 15.02.2016

doi 10.15802/stp2016/61003