УДК 004.40

ПАРАДИГМЫ ПРОГРАММИРОВАНИЯ СБОРОЧНОГО ТИПА В ПРОГРАММНОЙ ИНЖЕНРИИ

Е.М. Лаврищева

Киевский национальный университет имени Тараса Шевченко,
Киев, проспект Академика Глушкова, 4, email: lavryscheva@gmail.com

Представлен формальный аппарат парадигм программирования сборочного типа – модульного, объектного, компонентного и сервисного программирования. Теоретический аппарат каждой парадигмы обеспечивает формальную разработку отдельных программных элементов этих парадигм, определение интерфейсных элементов в стандартном виде и их сборку на единой технологической основе в среде комплекса ИТК.

A formal apparatus of paradigms of the programming assembling kind, such as modular, objective, component and service programming is presented: The theoretical apparatus of every paradigm secures the formal development of some program elements of these paradigms, determination of interface these elements in the standard kind and their assembling on the single technological basis in environment of complex of ITC IPS.

Вступление

Предлагаются теория парадигм программной инженерии и CASE-инструменты, предназначенные для разработки сложных программных систем из формализованных программных ресурсов методом сборки Разработан формальный аппарат каждой сборочной парадигмы программирования (модульной, объектной, компонентной, сервисный). Аппарат представлен теоретическими и прикладными аспектами проектирования отдельных соответствующих ресурсов в этих парадигмах и их сборку (конфигурацию) в сложные программные компьютерные системы. Представлен набор CASE-средств поддержки этих парадигм и проведения единообразного процесса изготовления отдельных ресурсов в каждой из парадигм и технология изготовления из них программных продуктов средствами сборочного конвейера. Формальные средства и CASE-инструменты их реализации созданы в рамках выполненных фундаментальных проектов ИПС НАНУ (2001–2011) по технологии программирования с участием научных специалистов отдела «Программная инженерия», аспирантов, докторантов и студентов Киевского национального университета имени Тараса Шевченко и филиала МФТИ в Институте кибернетики имени В.М. Глушкова НАН Украины.

В работе излагается формальный аппарат каждой их названных парадигм и общее описание системной поддержки процессов сборочного конвейера с помощью CASE–инструментов ИТК ИПС.

1. Парадигма модульного программирования

В 70-80 годы появилось модульное программирование, основу которого составляют методы декомпозиции предметной области и комплексирования модулей в сложные программные структуры. Модули реализуют функции и пользуются функциями других модулей. Они – самостоятельные и отдельные артефакты для некоторой предметной области (ПрО).

Модуль – это логически законченная часть программы, выполняющая определенную функцию. Он обладает такими свойствами: завершенность, независимость, повторное использование и др. Модули накапливались в библиотеках или Банках модулей, как готовые «детали», из которых собираются более сложные программные структуры и адаптируются к новым условиям среды их обработки.

Такие структуры проектировались методом снизу – вверх, выбирались готовые модули из Банков модулей и собирались в новые структуры. Этот метод был автоматизирован в рамках системы АПРОП в период (1976 – 1983 гг.) в Институте кибернетики АН УССР. В ней главными составляющими были: модули, интерфейсы и данные, которыми обменивались модули, метод сборки и функций преобразования типов данных между языками программирования (ЯП.) в классе языков (Фортран, ПЛ./1, Алгол, Ассемблер, Кобол) ОС ЕС [1–3].

Организация связи модулей через интерфейс. Каждая пара объединяемых разнородных модульных ресурсов связывается интерфейсом как модулем-посредником. Множество связываемых пар функциональных модулей в разных ЯП, модуля посредника в специальном языке объединяются в системе АПРОП в агрегатный продукт с четко выраженной модульной структурой на ЕС ЭВМ, отличающейся от традиционного структурного монолитного программирования.. В функции сборочного посредника входит передача данных между модулями через формальные и фактические параметры интерфейса и проверка соответствия их типов данных, количества и порядка расположения. Если типы данных параметров – нерелевантные (целое в вещественное), то в функции посредника входит прямое и обратное их преобразование. Сгенерированный модуль-посредник содержит обращения к элементам библиотеки интерфейса, которые выполняются в момент перехода от одного модуля к другому и обратно.

Фрагмент языка описания сборки модулей

<оператор связи>::= Link | Building |Cconfiguration < тип системы> < имя> (< список элементов>)

< тип системы> ::= < программная система>| < семейство программных систем>

< список элементов> : := < модуль> < список модулей>, < модуль>

< модуль> ::= < простой модуль> > | < интерфейсный модуль>

< простой модуль> :: = < имя модуля> | < сложное имя>| | < модуль>| | <заменяемое имя>| | < имя модуля>| < список ключевых параметров>| < сложное имя> | (< список ключевых параметров>) | <заменяемое имя> (< список ключевых параметров>)

< интерфейсный модуль> :: = < элемент связи > | < связанный модуль> & < элемент связи> | < элемент связи > | < связанный модуль>

< элемент связи > :: = < интерфейсный модуль> | < пусто>

< сложное имя> :: = < имя модуля> < имя точки входа>, < имя библиотеки>

< имя модуля> := < идентификатор> < пусто>

< имя точки входа> :: = < имя точки входа> < имя модуля> = < имя модуля> < имя модуля> < имя точки входа>| < имя модуля> . < имя точки входа>

< список ключевых параметров> : : = (< ключевой параметр>) | (< список ключевых параметров>, < ключевой параметр>)

< список ключевых параметров> :: = < формальный параметр модуля > = < фактический параметр>.

Функции системы автоматизации. Система АПРОП выполняет следующие функции объединения модулей:

– обработка паспортных (интерфейсных) данных модулей в ЯП;

– анализ описания параметров модулей и составление задания на обработку нерелевантных типов данных, проверка правильности передачи параметров по их количеству и по типам данных в классе ЯП;

– преобразование типов данных в ЯП (b-boolean, c-character, i-integer, r-real, a-array, z-record и др.) генератором типов данных с использованием функций библиотеки интерфейса;

– генерация модулей-посредников и составление таблицы связи пар компонентов;

– интеграция пар модулей и их размещение в структуре программного агрегата;

– трансляция и компиляция модулей в ЯП в виде готовой программной структуры;

– трассировка интерфейсов и отладка функций модулей в каждой паре агрегатной структуры;

– тестирование программного агрегата в целом;

– формирование программ запуска программного агрегата и документации.

В состав системы АПРОП входят следующие CASE-средства:

– системы программирования (ПЛ.1, Фортран, Ассемблер, ПЛ1, Фортран, язык модульного конструирования – ЯМК);

– элементы обработки (модули в ЯП, Банк модулей, Библиотека функций преобразования типов данных, интерпретатор языка, межмодульный интерфейс);

– средства автоматизации (обработка модулей в ЯП, генерация модулей связей, сборка разноязычных модулей, тестирование модулей, тестирования интерфейсов, тестирования сложного агрегата);

– модуль формирования выходного результата.

Таким образом, *интерфейс модулей* как средство связи разных типов объектов в ЯП – первая отечественная парадигма интерфейса в программировании реализована в системе АПРОП. Интерфейс развивался за рубежом в проектах MIL, SAA, OBERON для комплексирования модулей на других вычислительных системах.

Формальный аппарат парадигмы модульного программирования. Метод сборки включает математические формализмы определения связей (по данным и по управлению) между разноязычными модулями и генерации интерфейсных модулей-посредников для каждой пары разноязычных модулей [3–5].

Главная задача связи пары разноязычных модулей состоит в решении задачи построения взаимно однозначного соответствия между множеством фактических параметров $V = \{v^1, v^2, \ldots, v^k\}$ вызывающего модуля и множеством формальных параметров $F = \{f^1, f^2, \ldots, f^{k1}\}$ вызываемого модуля и отображения их данных с помощью алгебраических систем.

Каждому типу данных T_α^t языка 1_α поставлено в соответствие алгебраическая система вида:

$$G_\alpha^t =< X_\alpha^t, \Theta_\alpha^t >, \qquad (1)$$

где X_α^t – множество значений рассматриваемого типа, а Θ_α^t – множество операций над этим типом данных (ТД).

Сущность операции над ТД состоит в том, чтобы обеспечить преобразования одного типа к релевантному другому (например, real ↔ intger). Это необходимо тогда, когда некоторое значение данного передается через аппарат фактических параметров другому модулю и их ТД не совпадают.

То есть операции преобразования типов данных соответствует изоморфное отображение алгебраической системы G_α^t в G_β^d. Построенный класс алгебраических систем $\sum = \{G_\alpha^b, G_\alpha^c, G_\alpha^i, G_\alpha^r, G_\alpha^a, G_\alpha^z\}$ обеспечивает все виды преобразований типов данных (b-boolean, c-character, i-integer, r -real, a -array, z–record и др.) между собой. Каждой операции преобразования типов данных соответствует изоморфное отображение одной алгебраической системы в другую. Проведено доказательство изоморфизма отображения алгебраических систем для некоторых пар множеств X_α^t и X_β^q. При существовании изоморфизма мощности алгебраических систем равны $|G_\alpha^t| = |G_\beta^q|$.

Если отображение не удается выполнить, то задача автоматизированной связи для данной пары модулей считается неразрешимой.

Алгебраические системы построены в классе простых типов данных ЯП ($t = b, c, i, r$) и сложных типов данных ($t = a, z, u, e$) и допустимых видов операций их преобразования. Преобразования между ТД массивов и записей сводятся к определению изоморфизма между основными множествами соответствующих алгебраических систем с помощью операций изменения уровня структурирования данных – селектора и конструирования. Для массива операция селектора сводится к отображению множества индексов на множество значений элементов массива. Аналогично такая операция определяется для записи как отображение между селекторами компонентов записи и самими компонентами.

Формально преобразование неэквивалентных типов данных в ЯП выполняется следующими процедурами.

1. Построение операций преобразования типов данных $T = \{T_\alpha^t\}$ для множества языков программирования $L = \{l_\alpha\}_{\alpha=1,n}$.

2. Построение отображения простых типов данных для каждой пары взаимодействующих модулей в $l_{\alpha 1}$ и $l_{\alpha 2}$ и применение операций селектора *S* и конструктора *C* для отображения сложных структур данных в этих языках.

Формализованное преобразование типов данных осуществляется для каждого типа данных T_α^t:

$$G_\alpha^t = < X_\alpha^t, \Omega_\alpha^t >,$$

где *t* – тип данных; X_α^t – множество значений, которые могут принимать переменные этого типа; Ω_α^t – множество операций над этими типами данных.

Для простых и сложных типов данных ЯП построены следующие классы алгебраических систем:

$$\sum\nolimits_1 = \{G_\alpha^b, G_\alpha^c, G_\alpha^i, G_\alpha^r\},$$
$$\sum\nolimits_2 = \{G_\alpha^a, G_\alpha^z, G_\alpha^u, G_\alpha^e\}. \qquad (2)$$

Каждый элемент класса простых и сложных типов данных определяется на множестве их значений и операций над ними:

$$G_\alpha^t = < X_\alpha^t, \Omega_\alpha^t >, \qquad (3)$$

где $t = b, c, i, r, a, z, u, e$.

Операциям преобразования каждого *t* типа данных соответствует изоморфное отображение двух алгебраических систем с совместимыми типами данных двух разных ЯП. В классе систем $\sum_1$ и $\sum_2$ преобразование типов данных $t \to q$ для пары языков l_t и l_q определены следующие свойства отображений:

1) G_α^t и G_β^q – изоморфны (*q* определенный на том множестве, что и *t*);

2) между X_α^t и X_β^q существует изоморфизм, для которых множества $\Omega_\alpha{}^t$ и $\Omega_\beta{}^q$ разные. Если $\Omega = \Omega_\alpha{}^t \cap \cap \Omega_\beta{}^q$ не пусто, то рассматриваем изоморфизм между $G_\alpha^{t'} =< X_\alpha^t, \Omega >$ и $G_\beta^{q'} = < X_\beta^q, \Omega >$.

Такое преобразование сводится к первому случаю.

3) мощности алгебраических систем должны быть равны $|G_\alpha^t| = |G_\beta^q|$.

Любое отображение 1), 2) сохраняет линейный порядок, так как алгебраические системы (1) линейно упорядочены.

Между множествами X_α^t и X_β^q может не существовать изоморфного соответствия. В этом случае строится такое отображение между X_α^t и X_β^q, чтобы оно было изоморфным. Если такое отображение существует (в каждом конкретном случае оно может быть разным), то имеем условие случая 1) с соответствующими изменениями в определении алгебраических систем. Доказательство изоморфного отображения простых и структурных типов данных дано в [3–5].

Алгебра абстрактных ТД с операциями над ними реализована в рамках модульного программирования в системе АПРОП. В ней представлено 64 функции преобразования простых и сложных ТД для перечисленных ЯП. На основе описания интерфейса генерировался специальный модуль преобразования ТД. Библиотека функций интерфейса и генератор модулей связи между двумя разноязычными модулями были переданы в 52 организации страны по их заявкам.

Подобные функции реализованы на фирме IBM, Microsoft, Unix, Apple в составе их ОС. в виде библиотек реализации общих ТД для ЯП, а также модуля-посредника (stub, skeleton), параметры которого описываются в языке IDL (Interface Definition Language) и служат для обмена данными между объектами. Брокер объектных запросов (1994) выполняет функция взаимодействия разнородных объектов через интерфейс. Данные механизмы используются аналогично в объектной и компонентной парадигмах.

2. Парадигма объектного программирования

На рубеже 80-х XX столетия Г. Буч предложил объектный подход, который изменил сложившийся процесс структурного, функционального программирования, приведшего к *кризису сложности*. С этого момента развивалась *теория* Буча и *технология* выхода из кризиса.

Так появились объектно-ориентированные технология, ЯП, сформирован новый стиль программирования разных компьютерных систем путем моделирования ПрО объектами и методами. Появились объектно-ориентированные ЯП, библиотеки классов объектов, routines и ТД. Они стали ресурсом общего назначения для разработки многих сложных систем, автоматизированными системами ООП (COM, Corba, DCE RPC и т.п.).

Основу парадигмы объектного программирования составляет новый логико-математический аппарат четырехуровневого моделирования ПрО, разработанного в рамках докторской диссертации с.н.с В.Н. Грищенко На его основе моделировались функциональные и интерфейсные объекты при декомпозиции ПрО и последовательного их уточнения на уровнях проектирования в соответствии с концепцией логико-математического аппарата для каждого уровня. В результате, разработана технология построения ПС, в которой объединены на одной концептуальной основе теоретический аппарат объектного анализа и представления объектов-методов современными объектными ресурсами, их трансформации к программным компонентам в стандартной форме, принятой для модулей с целью их сборки в сложные структуры систем и семейств ПС путем конфигурирования вариантов ПС и СПС.

Далее дается описание теоретических и прикладных аспектов ОКМ, базовых понятий и положений парадигмы объектного программирования, включающей теории моделирования ПрО из объектов с доказательством изоморфизма отображения методов объектов в компоненты и их адаптация в разные репозитории современных сред функционирования [6–10].

Математическое моделирование объектной модели. Объектная теория построена с использованием базовых понятий Г. Буча (классы, полиморфизм, наследование и др.). Объект выделяется на уровнях объектного анализа с привлечением логико-математических формализмов для описания и уточнения функций объектов в объектной модели (ОМ) и отображения понятий, их сущностей, взаимоотношений и поведения объектов [6–8].

Объект – именуемая часть действительной реальности с определенным уровнем абстракции имеет денотат, знак и концепт. Каждый объект *OM* как сущность множества объектов $O = (O_0, O_1, .., O_n)$, где $O_1 = Oi(Nai, Deni, Coni)$, а $Nai, Deni, Coni$ соответственно означают – знак (имя), денотат и концепт объекта. $Coni = (P_{il}, P_{i2}, .., P_{is})$ определяется на множестве предикатов $P = (P_1, P_2, .., Pr)$.

Аксиома 1. Предметная область, моделирующаяся из объектов, сама является объектом.

Аксиома 2. Предметная область, что моделируется, может быть отдельным объектом в составе другой предметной области.

При моделировании объект ПрО имеет хотя бы одно свойство или характеристику и уникальную идентификацию в множестве объектов ПрО и множества предикатов свойств и отношений между ними.

Свойство объекта определяется на множестве объектов ПрО унарным предикатом, который принимает значение истины за его внешними и внутренними характеристиками.

Характеристика – это совокупность свойств (унарных предикатов) с условием получения значений истины не более чем одним предикатом из совокупности внешних и внутренних характеристик из области значений свойств, которое имеет истину.

Отношение определяется бинарным предикатом на множестве объектов ПрО, принимает значение истины на заданной паре объектов. Некоторые отношения имеют *роде-видовое* отношение (*IS-A*, либо *часть-целое* (*PART-OF*).

Уровни логико-математического моделирования ПрО. Проектирование модели ПрО выполняется на четырех уровнях логико-математического определения объектов предметной области:

1) **обобщающий уровень** определяет базовые понятия ПрО без учета их сущности и свойств;

2) **структурный уровень** определяет расположение объектов в структуре модели ПрО и установления отношений между ними;

3) **характеристический уровень** задает общие и специфические особенности концептов объектов;

4) **поведенческий уровень** определяет поведение и изменение объектов в зависимости от событий, которые они создают при их взаимодействии.

Каждому из четырех уровней проектирования ОМ соответствуют следующие концепции, описанные далее.

В соответствии с уровнем обобщения объект рассматривается как математическое понятие или класс, который можно трактоваться с точки зрения аксиоматической теории множества Геделя – Бернайса: множество $O = (O_0, O_1, .., O_n)$, в котором O_0 – объект ПрО.

На этом уровне формируется множество базовых функций ПрО путем декомпозиции или композиции денотатов и концептов объектов. Над множествами объектов могут выполняться базовые операции (объединения, пересечения, разницы, дополнения, принадлежности и др.).

Для множества объектов $O = (O_0, O_1, .., O_n)$ выполняется отношение:

$$\forall\, i\, [(i > 0) \& (O_i \in O_0)]. \tag{4}$$

При **структурном** уровни определяются такие понятия, как класс, экземпляр класса, абстрактный класс и др. Определение свойств объектов и их отношений (агрегация, детализация, классификация и др.) выполняется на характеристическом уровне, когда определяются: атрибут состояния, состояние, пространство состояний и др. Множество объектов упорядочивается и каждый из объектов может задаваться как множество или элемент множества. Тогда выражение (1) трансформируется в такой вид

$$\forall\, i\ \exists\, j\ [(i > 0)\ \&\ (j > 0)\ \&(i \neq j)\&(O_i \in O_j)], \tag{5}$$

который определяет отношение “часть–целое”.

В соответствии с **характеристическим** уровнем для каждого из объектов формируется соответствующий концепт. Если $O' = (O_1, O_2\ .\ O_n)$ – множество объектов ПрО, а $P' = (P_1, P_2\ .\ Pr)$ – множество унарных предикатов, связанных со свойствами объектов ПрО, и концепт объекта O_i определяется множеством утверждений, построенных на основе предикатов с P', которые принимают значение истины. То есть концепт $Coni = \{P_{ik}\}\}$ при условии $P_k(O_i) = true$, где P_{ik} является утверждением для объекта O_i в соответствии с предикатом P_k. Согласно этим правилам определяются свойства объектов в рамках данной абстракции и отношения “род–вид”.

Выражение $A = (O', P')$ определяет систему концептов объектов О’ алгебры и предикатов P' с помощью операций: *0-арных, унарных* и *бинарных*.

Аксиома 1. Каждый объект ПрО по крайней мере имеет хотя бы одну характеристику, которая определяет семантику и уникальную идентификацию в множестве объектов ПрО.

Исходя из **поведенческого** уровня определяется последовательность состояний объектов и их процессы с отображением переходов состояний. Взаимосвязи между объектами формируются на основе бинарных предикатов, которые связаны со свойствами объектов ПрО, и детализируются взаимосвязи между состояниями объектов.

Понятие класса объектов заменяется понятием множества. Если объект – элемент другого объекта, то он определяется классом. При этом не каждый объект является элементом любого другого объекта (класса). Конкретизация понятия объекта – это *класс*, который задает *экземпляры класса* – объект; *объединенный класс* – множество, которое является прямым произведением нескольких других экземпляров; *класс–пересечение* – это множество, которое является общей частью других множеств; *агрегированный класс* – это множество, которое является подмножеством определенного декартового произведения нескольких других множеств объектов.

Алгебра объектного анализа ПрО

Алгебра включает объекты и операции над ними:

$$\sum = (O', I', A'), \tag{6}$$

где $O' = (O_1, O_2, \ldots, O_n)$ – множество объектов, $I = (I_1, I_2, \ldots, In)$ – множество интерфейсов; $A' = (A_1, A_2, \ldots$ $\ldots, A_n)$ – множество (Action – *A’*) операций над элементами множества *O*. Каждая из операций *A’* имеет приоритет и арность, а также связанные с соответствующими допустимыми описаниями концептов объектов и операциями множества *A’ = {decds, decdn, comds, comdn, conexp, connar}*, где *decds, decdn* – декомпозиции, *comds, comdn* – композиции и *conexp, connar* – сужение.

Модель ПрО задается объектным графом G Граф соответствует обобщенному и структурному уровню логико-математического проектирования ОМ, включающих интерфейсные объекты для их связи между собой. Граф $G=\{O, I, R\}$ определен на множестве объектов O, интерфейсов I и отношений R (relations) между объектами.

Вершины графа G залают объекты, которые находятся в репозитории, а дуги соответствуют отношениям между ними, Отношения могут задаваться интерфейсными объектами

Элементы графа описываются в ЯП, а интерфейсные объекты в специальном языке системы Corba. Параметры внешних характеристик интерфейсных объектов передаются между объектами и специфицируются *in, out, inout* в языке IDL системы Corba. Объекты помещаются в библиотеку или репозиторий.

Таким образом, модель ПрО задается объектным графом $G=\{O, I, R\}$, определенным на множестве объектов ПрО, интерфейсов I и отношений R (relations) между объектами. Граф имеет такие правила:

– существует хотя бы одна вершина, имеющая статус множество–объект и отображающая ПрО в целом;

– множество вершин *графа*, задающее взаимно однозначное отображение объектов ПрО;

– каждой вершине соответствует хотя бы один интерфейс $Ik \in I$ и отношения, которые принадлежат множеству R соответственно правил.

Множество объектов–функций связано с набором методов реализации удаленных объектов ПрО. При конкретизации объекты графа G имеют связи через интерфейсные объекты из множества I. То есть вершины графа G – объекты двух типов – функциональные O и интерфейсные I.

Интерфейсным объектам графа соответствует описание данных, методы их передачи через запросы RPC или RMI данных и возможные операции преобразования этих данных к соответствующим форматам среды выполнения.

Результат связи двух объектов графа (например, O25, O47) – это интерфейсный объект, в которого множество входных интерфейсов совпадает с множеством интерфейсов объекта-приемника, а множество исходных интерфейсов с множеством исходных интерфейсов объекта-передатчика.

Аксиома 4. Построенный граф G дополненный интерфейсными объектами структурно, упорядочивающийся (наверх) с контролем полноты, избыточности и устранения дублирующих элементов.

Объекты могут иметь несколько интерфейсов, которые могут наследовать интерфейсы других объектов, тогда последние предоставляют сервис всего множества исходных интерфейсов.

Множество объектов и интерфейсов графа задается общими и индивидуальными характеристиками ОМ. Каждая из них это попарно сравнение внутренних характеристик объекта с их внешними характеристиками. Они считаются достоверными, если выполняются такие условия: каждое внутреннее свойство эквивалентно внешнему свойству объекта. Если это условие не выполняется, то такой элемент удаляется из списка элементов множества и из графа соответственно.

Формальные механизмы определения ПС с помощью объектов и интерфейсов

Базовые основы объектного проектирования:

F – множество функций объектов

O – множество объектов $O=O_0, O_2, \ldots, O_k$;

I – множество интерфейсов *In, Out,* в котором *In* – множество входных интерфейсных объектов, в которых описываются данные клиента для передаче их серверному объекту $O_k \in O$, $In(O_k)$ и *Out* – множество выходных параметров из интерфейсов от сервера $Ok \in O$, *Out* (*Ok*) и *Inout* – промежуточные интерфейсы.

Модель объединения объектов основывается на свойствах и характеристиках объектов модели ОМ, которые отображаются в описании интерфейса компонентов в языке IDL (параметры *In, Out*) и с помощью операций принадлежности. Результатом объединения двух объектов будет объект, у которого множество входных интерфейсов ($O_k \leftarrow O_l$) совпадают с множеством выходных интерфейсов объекта от клиента, а множество выходных интерфейсов совпадает с множеством входных интерфейсов объекта от сервера:

Аксиома 5. Операция взаимодействия O_k, O_l дает объект, в которого множество интерфейсов *In* совпадает с множеством интерфейсов *Out*, а множество *Out* интерфейсов с множеством *In* интерфейсов:

$$O_k = (Out(O_k), In(O_k)),$$

$$O_l = (Out(O_l), In(O_l)),$$

$$O_k \cdot O_l = (Out(O_k), In(O_l)).$$

Аксиома 6. Композиция объектов $O_k \cdot O_l$ является корректной, если объект–клиент полностью обеспечивает сервис, необходимый объекту–серверу, то есть:

$$\forall I_m \in In(O_k) \Rightarrow \exists I_n \in Out(O_l) \wedge I_m = I_n.$$

Компонентные объекты могут иметь несколько интерфейсов, которые могут наследовать интерфейсы других объектов ($O_k \leftarrow O_l$), тогда последние предоставляют сервис для всего множества входных интерфейсов: $O_k \leftarrow O_l \Rightarrow Out(O_k) \subseteq Out(O_l)$.

В случае, когда объект наследует другой объект, у которого множество входных интерфейсов содержит все его интерфейсы, а множество выходных интерфейсов содержит только интерфейсы, которые необходимы для задания сервиса.

Формальные операции над объектами и интерфейсами. Рассматривается несколько проекций объектов:

– проекция объекта на *интерфейс* как объект, в котором множество интерфейсов *In* содержит один входной интерфейс, а множество интерфейсов *Out* – содержит только те интерфейсы, которые необходимы для задания сервиса;.

– проекция *объекта на объект* , как проекция объекта на множестве входных интерфейсов объекта;

– проекция *объекта на объект с взаимодействием* обеспечивается равенством:

$$(O_k \cdot O_l)[I_m] = O_k[I_m] \cdot O_l.$$

К формальным операциям относятся:

– операция *параллельного выполнения РПС*.

$$\text{Po} \parallel . \parallel \text{Pr}$$

– операция *взаимодействия объектов и среды* задает объект, в котором множество интерфейсов *In* совпадает с множеством интерфейсов *In* объекта-сервера, а множество интерфейсов *Out* обеспечивает объединение множество интерфейсов *Out* объектов из среды.

Аксиома 7. Взаимодействие объекта и среды является корректным, если выполняется условие: среда полностью обеспечивает сервис, необходимый объекту клиента

$$O_k \cdot (O_{l_1} || \ldots || O_{l_n}) = \left(Out(O_k), \bigcup_{j=1}^{n} In(O_{l_j}) \right),$$

$$\forall I_m \in In(O_k) \Rightarrow \exists I_n \in \bigcup_{j=1}^{n} Out(O_{l_j}) \wedge I_m = I_n.$$

Операция наследования объектом интерфейса из РПС дает объект, который унаследовал интерфейсы всех объектов среды. Объект, который наследуется, передает все интерфейсы и имеет следующие свойства:

транзитивности $\forall O_{1,2,3} \in O : O_1 \leftarrow O_2, O_2 \leftarrow O_3 \Rightarrow O_1 \leftarrow O_3$;

симметричности $\forall O_k \in O \Rightarrow O_k \leftarrow O_k$.

Операция параллельного выполнения программ РПС не всегда является симметричной:

$$O_k \leftarrow (O_{l_1} || O_{l_2}) \neq O_k \leftarrow (O_{l_2} || O_{l_1}).$$

Результатом отображения объекта на объект является интерфейс из множества входных интерфейсов $O_k[O_l] = O_k[In(O_l)]$ и выходных интерфейсов $O_k[O_l] = Ok\ [Out(O_i)]$.

Спроектированные объекты могут трансформироваться к программным компонентам, которые сертифицируются и накапливаются в репозитории комплекса ИТК для последующего использования.

3. Парадигма компонетного программирования

Одно из ключевых направлений в компонентном программировании – это построение компонентов повторного использования (ПИК), которыми могут быть программы и модули, пригодные для использования в новой разработке ПС. В ПИК материализован многолетний опыт компьютеризации разных сфер человеческой деятельности, который может использоваться непосредственно адаптироваться к новым условиям среды применения.

Программирование с использованием ПИК обогатило метод проектирования (снизу-вверх) сложных программ с более простых. Оно уменьшает затраты на разработку за счет выбора ПИК с типовыми функциями и настройки их на новые условия применения, на что тратится гораздо меньше усилий, чем на аналогичную разработку. Компонент конструируется как некоторая абстракция, реализующая некоторую функцию предметной области и описывается в любом ЯП и не зависит от операционной среды и от реальной платформы.

Компонентное программирование – природное расширение ООП. Компоненты не являются объектами, а предоставляют им необходимые ресурсы и сервис. Сформировались понятия и положения о компонентах подходы к спецификации компонентов и их интерфейсов, способах сборки и композиции, а также отдельные теоретические положения парадигмы компонентного программирования. В ней заложена фундаментальная идея

композиции и сборки компонентов. Это стиль программирования, имеет свою концептуальную базу, теорию, методологию и инструменты.

В рамках компонентного программирования разработан новый метод ОКМ комплексного анализа и компонентного построения сложных программных систем. В нем дано обобщение понятия объектов, как элементов реального мира с соответствующими свойствами и характеристиками, которые последовательно определяются и уточняются с помощью логико-математических формализмов представления объектов, с присвоением им необходимых и достаточных свойств и характеристик, которыми они отличаются друг от друга. Метод объектного анализа дает отображение объектов в программные компоненты и интерфейсы, обеспечивает последовательного переход от объектов к компонентам и их интерфейсам [6–10].

В компонентном программировании определен метод сборки программных элементов и разработан формальный и математический аппарат, который включает модели компонента, компонентной среды и компонентной программы, внешнюю и внутреннюю компонентные алгебры, а также формализмы описания интерфейсов и выполнения системы преобразования ТД на основе заданной сигнатуры интерфейсов взаимодействующих компонентов в ПС [5, 9 –11].

Определен формальный механизм перехода от объектов к компонентам и их интерфейсам, а также дано формальное определение КПИ.

$$\text{КПИ} = (T, I, F, R, S),$$

где T – тип компонента, I – интерфейс компонента; F – функциональность, R – реализация, S – сервис взаимосвязи с компонентами и средой [8].

Кроме того определен набор формальных моделей (*компонента, интерфейса, компонентной среды),* операций компонентной алгебры (внешний и внутренний) и усовершенствована теория преобразования несовместимых типов данных компонентов FDT в ЯП.

Модель компонента – следствие обобщенных типовых решений относительно сущности компонента, его архитектуры, структуры, свойств, характеристик и др. Формально модель компонента имеет вид:

$$Comp = (CName, CInt, CFact, CImp, CServ), \tag{7}$$

где $CName$ – уникальное имя компонента;

$CInt = \{CInt^i\}$ – множество интерфейсов, связанных с компонентом;

$CFact$ – интерфейс управления экземплярами компонента;

$CImp = \{CImp^j\}$ – множество реализаций компонента;

$CServ = \{CServ^r\}$ – множество системных сервисов.

Возможны отношения между объектами типа: наследование, экземпляризация, контракт, связывание и взаимодействие.

Модель компонентной среды:

$$CE = (NameSpace, IntRep, ImpRep, CServ, CServ\,Imp),$$

где $NameSpace = \{CName^m\}$ – множество имен компонентов среды;

$IntRep = \{IntRep^i\}$ – репозиторий интерфейсов компонентов среды;

$ImpRep = \{ImpRep^j\}$ – репозиторий реализаций.

$CServ = \{CServ^r\}$ – интерфейс множества системных сервисов;

$CServImp = \{CServImp^r\}$ – множество реализаций для системных сервисов.

Модель интерфейса. Каждый *i–интерфейс* компонента имеет вид

$$CInt^i = \{IntName^i, IntFunc^i, IntSpec^i\}, \tag{8}$$

где $IntName^i$ – имя интерфейса; $IntFunc^i$ – функциональность, реализованная данным интерфейсом (совокупность методов); $CInt^i$ – интерфейс управления экземплярами компонента; $IntSpec^i$ – спецификация интерфейса (описания типов, констант, других элементов данных, сигнатур методов и т. д.).

Необходимым требованием существования компонента является условие его целостности:

$$\forall CInt^i \in CInt\ \exists CImp^j \in CImp[Provide(CInt^i \subseteq CImp^j)], \tag{9}$$

где $Provide(CInt^i)$ означает функциональность, которая обеспечивает реализацию методов интерфейса $CInt^i$.

Наличие знака включения в данной формуле означает, что избранная реализация компонента может обеспечить поддержку не только необходимого интерфейса, но и других возможностей. Для этого практические технологии и ЯП (CORBA, Java, C++ и др.) содержат необходимые средства. Интерфейс может иметь

несколько реализаций, которые различаются особенностями функционирования (например, операционной средой, средствами сохранения данных и т. д.).

Взаимодействие двух компонентов $Comp_1$ и $Comp_2$ определяется следующим необходимым условием: если $CInt^i_1 \in CInt_1$, то должен существовать $CInt^k_2 \in CInt_2$ такой, что

$$Sign(CInt^i_1) = Sign(CInt^k_2) \& Provide(CInt^i_1) \subseteq CImp^j_2,$$

где $Sign(\ldots)$ означает сигнатуру соответствующего интерфейса.

Пары компонентов компонентной модели определенного типа ($CFact$ и $CServ$ – фиксированные) при сопоставлении имеют следующие свойства.

Условия распределенной среды и проблемы возникновения и устранения угроз компонентам и каркасам ставят перед разработчиками ПС задачу управления атрибутами качества, безопасности и др. для обеспечения защиты компонентов от разрушений и выходов из тупика.

Для трансформации моделей к выходному коду разработана компонентная алгебра, включающая в себя внешнюю, внутреннюю и эволюционную алгебры.

Внешняя компонентная алгебра построена из следующих элементов:

$$\varphi_1 = \{CSet, CESet, \Omega_1\},$$

где $CSet$ – множество компонентов; $CESet$ – множество компонентных сред; Ω_1 – множество операций над этими элементами. Операции такие :

– инсталляция (развертывание компонента) $CSet_2 = Cset \oplus CESet_1$;

– объединение компонентных сред $CESet_3 = CESet_1 \cup CESet_2$;

– удаление компонента из компонентной среды $CESet_2 = CESet_1 \setminus CSet$.

Внутренняя компонентная алгебра имеет вид:

$$\varphi_2 = \{CSet, CESet, \Omega_2\},$$

где $Cset = \{OldComp, NewComp\}$ – множества старых *OldComp* компонентов в системе и множество новых компонентов *NewComp*, вновь разработанных или некоторого преобразованного старого компонента к новому *NewComp;*

OldComp = (*OldCName, OldInt, CFact, OldImp, CServ*) включает интерфейсы, реализации в серверной среде;

NewComp = (*NewCName, NewInt, CFact, NewImp, CServ*) включает в себя интерфейсы, реализации этого компонента, как необходимые элементы любого компонента, том числе и нового компонента в серверной среде;

$$\Omega_2 = \{addImp, addInt, replInt, replImp\},$$

где *add imp* – операции добавления реализации; *addInt* – добавления интерфейса; *replImp* – операция замещения реализации компонента, *replImp* – замещения интерфейса компонента.

Алгебра эволюции φ_3 включает в себя операции

$$\Omega_3 = \{ O_{refac}, O_{Reing}, O_{Rever} \},$$

где O_{refac} – рефакторинга, O_{Reing} – реинженерии и O_{Rever} – реверсной инженерии компонентов. Семантика этих операций такая:

рефакторинг обеспечивает построение компонентной ПСс возможностью внесения изменений;

реинженерия и реверсная инженерия обеспечивают замену, переименование компонентов и/или их интерфейсов и добавление новых компонентов в ПС, компонентная среда которой дополняется новыми или измененными компонентами.

Операции данной алгебры обеспечивают изменение компонентов и интерфейсов с помощью специальных операций, определенных в соответствующих моделях.

Модель рефакторинга компонентов:

$$M_{refac} = \{O_{кefac}, \{CSet = \{NewComp^n\}\},$$

где $O_{refac} = \{AddOImp, AddNImp, ReplImp, AddInt\}$ – операции рефакторинга, пара $(CSet, O_{refac})$ – элемент компонентной алгебры эволюции.

Операция рефакторинга носит комплексный характер, так как методы, входящие в этот интерфейс, требуют реализации компонента в составе контейнера. Поэтому реализация этой операции связана с существо-

ванием нового типа контейнера. Согласно классификации методов рефакторинга, операция расширения интерфейса управляет экземплярами компонентов и относится к расширенной классификации.

Множество операций рефакторинга *AddOImp, AddNImp, ReplImp, AddInt* включают в себя операции добавления и замещения реализаций компонентов и интерфейсов. Другими словами, пара (*CSet, Refac*) входит в компонентную алгебру и включает в себя методы обеспечения рефакторинга.

Таким образом, компонентная алгебра имеет вид**:**

$$\sum = \{\varphi_1, \varphi_2, \varphi_3\},$$

где $\varphi_1 = \{CSet, CESet, \Omega_1\}$ – внешняя алгебра, $\varphi_2 = \{CSet, CESet, \Omega_2\}$ – внутренняя алгебра, $\varphi_3 = \{Set, CESet, \Omega_3\}$ – алгебра эволюции компонентов.

4. Сервисное программирование

История развития программирования с самого начала связана с использованием сервисов и услуг. Сервисы реализует некоторые дополнительные возможности, которые необходимы многим программистам и потенциальным пользователям [15–17]. Как правило, описания сервисов содержат в себе информацию назначении и форме представления. Сформировалось три вида сервисов [11–14]**.**

– общие системные, которые есть в каждой общесистемной среде для поддержки процессов проектирования и реализации ПС;

– объектные сервисы, поддерживающие объекты и классы, операции ЖЦ, услуги необходимы для разработки объектно-ориентированных систем;

– веб-сервисы, которые являются информационными ресурсами Интернет, предоставляющие бизнес услуги для решения задач в вебах всемерной паутины Интернет.

Каждый сервис имеет: имя (Name), способствующее поиску в распределенной среде пространства имен: связь Binding для задания соответствия “имя-объект”; транзакция (transaction) для организации и управления отдельными компонентами в Интернет; сообщение (Messaging) для общения с компонентами.

Перечисленные четыре вида сервисов обязательные для любой модели ПС и ее реализации. С их помощью осуществляется: поиск компонентов; доступ к их ресурсам; организация обмена информацией между компонентами; динамическое управление функциями, обусловленными некоторыми готовыми КПИ в системе Интернет.

С общей точки зрения модель сервиса имеет типизированную структуру и интерфейс. Для представления разных аспектов описания и функционирования сервисов используется язык XML. К этим аспектам относятся описание структур и семантики данных, механизмов взаимодействия с сервисами, функциональных услуг и механизм поиска необходимых сервисов.

Средством описания механизма взаимодействия с сервисами является SOAP, а для описания функциональности сервисов – язык WSDL. Описание функциональности сервисов выполняется унифицированными языками XML, WSDL; описание структуры и семантики данных – RDF; описание процессов представления и обработки сервисов языком BPMN и взаимодействия с сервисами и поиска необходимых сервисов – язык SOAP.

Основная форма реализации сервисов – это *веб-сервисы,* которые сохраняются и идентифицируются URL–адресами и взаимодействуют между собой посредством сообщений через сеть Интернета. Веб–сервис имеет URL–адрес, интерфейс и механизм взаимодействия с другим сервисом через протоколы Интернета. Обмен данными между веб-сервисом и программой осуществляется с помощью XML-документов, оформленных в виде сообщений. Веб-сервисы могут способствовать решению бизнес задач разной природы через провайдеры сети Интернета.

Основные средства описания и разработки новых систем средствами веб-сервисов:

– язык XML для описания и построения SOA-архитектуры;

– язык WSDL (Web Services Description Language) для описания веб-ервисов и их интерфейсов в XML, что касается типов данных и сообщений, а также моделей взаимодействия и протоколов связи сервисов между собой;

– SOAP (Simple Object Access Protocol) для определения форматов запросов к веб-сервисам;

– SCA (Servise-Component Architecture) для создания более сложной системы на основе компонентов и сервисов;

– UDDI (Universal Description, Discovery and Integration) для универсального описания, выявления и интеграции сервисов, обеспечения их хранения, упорядочения деловой сервисной информации в специальном реестре с указателями на конкретные интерфейсы веб-сервисов.

Подходом к использования сервисов для моделирования сложных систем и решения разного рода задач на основе моделей SOA и **SCA.**

Базовые модели сервисов SOA и SCA. Модель SOA (Servise Oriented Architecture). Объект сервисно-ориентированной архитектуры – это веб-сервис, применяющий две технологии, что обеспечивают функциональность (Functions) и качество сервисов (Quality service). Эти технологии вынесены на уровень IT-стандартов комитета W3C и имеют следующие уровни.

Технология обеспечения функциональности веб-сервисов включает:

– транспортный уровень (transport layer) для обмена данными;

– коммуникационный уровень (service communication layer) для определения протоколов;

– уровень описания сервиса (service description layer) и связанных с ним интерфейсов;

– уровень бизнес-процессов (business process layer) для реализации бизнес-процессов и потоков работ через механизмы веб-сервисов;

– уровень реестра сервисов (service registry layer), который обеспечивает организацию библиотек веб-сервисов для их публикации, поиска и вызова за их WSDL–описаниями интерфейсов. Технология обеспечения качества веб-сервисов имеет следующие уровни:

– политики (policy layer) для описания правил и условий применения веб-сервисов;

– безопасности (security layer) для описания вопросов безопасности веб-сервисов и функционирования (авторизация, аутентификация и распределение доступа);

– транзакций (transaction layer) для установления параметров обращения к веб-сервисам и обеспечению надежности их функционирования;

– управление (management layer) веб-сервисами.

Технологический фундамент веб-сервисов составляют: XML, SOAP, UDDI, WSDL. С их помощью осуществляется реализация базовых свойств веб-сервиса и механизма взаимодействия между собой веб-сервисов в среде SOA;

– провайдер сервиса, который осуществляет реализацию сервиса в виде веб-сервиса, прием и выполнение запросов пользователей сервиса, а также публикацию сервиса, отмеченного в реестре сервисов;

– реестр сервисов, которой содержит в себе библиотеку сервисов для пользователей сервиса и средства поиска и вызова необходимого сервиса за запросами, которые поступили от провайдеров сервисов на предоставление сервисов;

– пользователь сервиса (приложение, программный модуль или сервис), который осуществляет поиск и вызов необходимого сервиса из реестра описания сервисов, а также использует сервис, предоставленный провайдером в соответствии его интерфейсу.

Модель SCA (Servise-Component Architecture) обеспечивает доступ к сервисным компонентам и определить зависимости между ними через аппарат ссылок. Компоненты SCA системы IBM ® WebSphere ® Integration Developer могут быть упакованы в модуль для выполнения сервисного модуля с WebSphere Process Server – эквивалентного EAR-файлу J2EE и некоторым другим. Подмодули J2EE и артефакты упаковываются с модулем SCA Это позволяет запустить сервис через модель SCA и , передавать данные при интеграции. Механизмы, которые используются для вызова внешнего сервиса, названного *импортом* и *экспортом*. Они связаны с другими технологиями, таких как JMS, Enterprise JavaBeans или Web-сервисы. SCA модуль позволяет обратиться к существующему Enterprise JavaBean, используя модель SCA. Элементы SCA могут компоноваться и обмениваться данными друг с другом, пересылая SDO, подготовленные в необходимом виде. Этот интерфейс включает определения метода получения и установления свойства данных. WebSphere Integration Developer инструментарий для разработки применения на платформе Eclipse.

Принцип реализации веб-сервисов в ИТК. Веб–сервис В ИТК идентифицируется программой (сложение и вычитание чисел) из рядков URI, свойства и методы которой описаны с использованием языка WSDL. Доступ к ресурсам осуществляется через протокол SOAP, который представляется XML-запросами, передаваемые HTTP Интернет–протоколом. Веб-сервисы близки классам Java EE. Ключевым понятием веб-сервиса является сообщение из одной или нескольких переменных. Методы классов задаются операциями с входным и выходным значениями сообщений. После вызова операции переменной входного сообщения протокола SOAP, интерпретируются как параметры соответствующего метода класса, который лежит в основе сервиса. После завершения работы метода формируется исходное сообщение, которое содержит возвращенное методом значение, после чего отправляется клиенту протоколом SOAP.

5. Технология реализация отдельных задач парадигм програмирования

Методология проектирования и реализации ПС методом сборочного программирования из *готовых элементов парадигм* с помощью CASE-инструментов (трансляторы с ЯП, тестирование, трасформеры, генераторы и т. п.), а также инструментальных средств (Eclipse, Protege, VS.Net, Corba, Java и т.п) реализованы в инструментально–технологического комплексе (ИТК) [17–23]. В нем готовым ресурсом от парадигм является КПИ (reuse, assets, artifacts, services и т. п.). Они отображают некоторые функций ПрО. Каждый КПИ специфицируется соответствующими стандартами путем описания данных в языке WSDL, а интерфейс в языках IDL, API, SDIL и др. Это дает возможность собирать КПИ на единой основе, общей для всех видов разнородных ресурсов [15–19].

Технология проектирования ПС и СПС из готовых ресурсов включает:

– проектирования ПС с использованием процессов стандартного ЖЦ;

– проектирование ПС и СПС с помощью моделей MDD, MDA;

– онтологическое проектирование доменов с заданием модели характеристик сущностей модели доменов и доведения ее к созданию архитектуры системы из готовых компонентов средствами сборочного конвейера;

– спецификация разнородных программных ресурсов в ЯП, их реализация, тестирование для проверки правильности и накопления верифицированного компонента в репозитории системы вместе с паспортом;

– отбор готовых компонентов в репозитории средствами созданной фабрики программ;

– сбор разнородных КПИ новых ПС для реализации некоторых задач автоматизируемой предметной области;

– генерация из некоторых асетов с артефактов исходного кода и адаптации их под конкретные цели ранее созданного программного решения или программы;

– трансформация с описанием специфики ПрО графическим языком DSL и использованием DSL TooLs VS.Net для получения исходного кода построенной объектно- компонентной модели;

– тестирование КПИ и ПС со сбором необходимых данных для оценки качества ПС;

– инженерия качества программных систем, включая экспертный и метрический анализ показателей качества и достижения показателей качества ПП;

– сохранение результатов проектирования в репозитории компонентов;

– документирование КПИ, новых компонентов в составе домену.

В комплексе ИТК реализованы основные, общие элементы парадигм программирования, необходимые при проектирования доменов, ПС из КПИ. В нем нашли представление фундаментальных положений парадигм, включая; теории взаимодействия и вариантности ПС; теория моделирования и адаптации спроектированных ПС в других средах, технология изготовления ПС по 10 линиям из КПИ; процессы стандарта ЖЦ 12207 и оценки качества ISO/IEC 3226, 9000 (1–4), онтология вычислительной геометрии; линия обучения современным языкам C#, Java и CASE-инструментам – Protégé, Eclipse, VS.Net, Java и др.

К технологическим линиям можно обращаться через веб-сайт (http://sestudy.edu-ua.net). К линии обучения предмета «Программной инженерии» осуществляется экспериментальной фабрикой программ сайта http://programsfactory.univ.kiev.ua), изготовленного студентами (Ароновым и А. Дзюбенко) под руководством автора. Этому курсу можно обучаться и на сайте www.intuit.ru.

1. *Глушков В.М., Лаврищева Е.М., Стогний А.А. и др.* Система автоматизации производства программ (АПРОП). – К.: ИК АН УССР, 1976. – 134 с.
2. *Лаврищева Е.М.* Становление и развитие модульно-компонентной инженерии программирования в Украине // Препринт 2008–1. – Ин-т кибернетики им. В.М. Глушкова, – 33 с.
3. *Лаврищева Е.М.* Методы программирования. Теория, инженерия, практика. – К.: Наук. думка, 2006.– 451 с. (www.twirpx.com/
4. *Лаврищева Е.М., Грищенко В.Н.* Сборочное программирование. – К.: Наук. думка, 1991 –213 с.
5. *Лаврищева Е.М., Грищенко В.Н.* Сборочное программирование. Основы индустрии программных продуктов. – К.: Наук. думка, 2009. – 371 с.
6. *Грищенко В.Н.* Метод об'єктно-компонентного проектування програмних систем // Проблеми програмування. – 2007. – № 2. – С. 113–125.
7. *Лаврищева Е.М., Грищенко В.Н.* Методы и средства компонентного программирования // Кибернетика и системный анализ. – 2003.– № 1. – С. 39–55.
8. *Лаврищева К.М.* Компонентне програмування. Теорія і практика // Проблеми програмування. – 2010. – № 1. – С. 17–29.
9. *Лавріщева К.М., Колесник А.Л., Стеняшин А.Ю.* Об'єктне-компонентне проектування програмних систем. Теоретичні і прикладні питання. – Вісник КГУ, серія фіз.-мат. наук. – 2013. – № 4. – С. 150–164.
10. *Лавріщева К.М., Коваль Г.І., Бабенко Л.П. та ін.* Нові теоретичні засади технології виробництва сімейств програмних систем у контексті ГП. – Електронна монографія, ДРНТІ. – № 67–УК–2011 від 05.10.11. – 377 с.
11. *Лаврищева Е.М.* Концепція індустрії паукового софтвера і підхід до обчислення наукових задач // Проблеми програмування. – 2011. – № 1.– С. 3–17.
12. *Лаврішева К.*М. Взаємодія програм, систем й операційних середовищ // Проблеми програмування. – 2011. – № 3. – С. 13–24.
13. *Аронов А.О. Дзюбенко А.І.* Підхід до створення студентської фабрики программ // Проблеми програмування. – 2011. – № 3. – С. 42–49.
14. *Лавріщева К.М.* Програмна інженерія.– Подручник.– Академперіодика, 2008. – 319 с.
15. *Lavrischeva E., Aronov A., Dzubenko A.* Programs factory – a conception of Knowledge Representation of Scientifical Standpoint of Software Engineering // Jornal of Computer Science, Canadian Senter of Science and Education. – 2013. – P. 21–27.
16. *Лавріщева Е.М., Зінькович В.М., Колесник А.Л. та ін.* Інструментально-технологичний комплекс розробки и навчання прийомам виробництва програмних систем. – Державна служба інтелектуальної власності України. – Свідоцтво про реєстрацію авторського права на твір. – № 45292, від 27.08.2012.
17. *Лаврищева К.М.* Розвиток идей академіка В.М. Глушкова з питань технологии програмування. – К.: Вісник НАН України. – 2013. – № 9. – С. 66–83.
18. *Lavrischeva E., Dzubenko A., Aronov A.* Conception of Programs factory for Representation and E-learning Disciplines of Software Engineering // 9– th International Conf. ICTERI– 2013 "ICT in Education, Research and Industrial Applications; Integration, Harmonization and Knowledge Transfer", Ukraine, June 17–21, 2013 http://ceur-ws.org/Vol-1000/
19. *Лаврищева Е.М.* Сборочный конвейер фабрик программ – идея академика Глушкова // В.М.Глушков. Прошлое устремленное в будущее. – Киев: Академпериодика, 2013. – С. 130–139.