ПРОГРАММНО–
ТЕХНИЧЕСКИЕ
КОМПЛЕКСЫ

Е.М. ЛАВРИЩЕВА

УДК 681.3.06

ФОРМАЛЬНЫЕ ОСНОВЫ ИНТЕРОПЕРАБЕЛЬНОСТИ КОМПОНЕНТОВ В ПРОГРАММИРОВАНИИ

Ключевые слова: *интероперабельность, компонент, повторное использование, компонентное программирование, изоморфизм, преобразование, фундаментальные типы данных, общие типы данных, гетерогенная среда*.

ВВЕДЕНИЕ

Возможность взаимодействия двух и более компонентов программ и систем в целях обмена информацией и использования ее для организации вычислений определяется как интероперабельность [1, 2]. Способность программ к взаимодействию заложена в языках программирования (ЯП), реализована в современных системах и средах общего назначения. К механизмам обеспечения интероперабельности относятся: аппарат подпрограмм и функций в ЯП; интерфейс в разноязыковых и разноплатформенных программах через интерфейсный-посредник в языках типа MIL (Module Interconnection Language), IDL (Interface Definition Language) и другие; брокер запросов по взаимодействию разноязыковых программ в системах общего назначения (CORBA, COM, Sun IBM, JAVA и другие); конфигурация и интеграция на уровне выходного кода программ в системах Linux, Windows Server, Microsoft.Net, IBM Web Sphere и т.п.

Базисом взаимодействия компонентов, программ и систем являются фундаментальные типы данных (ФТД), используемые при описании простых и структурных данных программ, значения которых передаются с помощью операторов вызова, запросов или сетевых протоколов другим программным объектам. Понятие интероперабельности появилось в связи с необходимостью практического объединения модулей, компонентов и программ. В настоящее время оно в большей степени предполагает использование готовых программных объектов в процессе разработки новых программных средств (ПС) (общесистемных, прикладных, проблемных, научных и других), с помощью которых решаются разного рода задачи в вычислительных средах. Обеспечение интероперабельности связано со многими факторами: совместимостью платформ компьютеров, структур и типов данных программных объектов, описанных в ЯП [3–8], и возможностями современных сред. Этот процесс осложняется реально существующими различиями в выходном коде систем программирования с ЯП, конфигурационном файле готовых ПС, а также в принципах обеспечения механизмов взаимодействия в вычислительных или гетерогенных средах.

Разработка новых формализмов взаимодействия стимулируется расширением спектра технических средств (мейнфреймы, кластеры, гриды и другие), структур данных (графы, скрипты, медийные объекты и другие) [9], языков описания инфор-

мационных ресурсов (XML, RDF, OWL и другие), а также стандартами описания и генерации общих типов данных (ISO/IEC 11404:2007. General Data Types), которые в перспективе должны улучшить процесс спецификации интерфейсов разнородных объектов и их выполнения в гетерогенных средах.

Основой обеспечения интероперабельности являются хорошо зарекомендовавшая себя теория структурной организации и типов данных ЯП [4–9], языки спецификации интерфейсов компонентов, методы композиции компонентов с помощью функций релевантного отображения типов передаваемых данных между компонентами и новые принципы организации вычислений в гетерогенных средах [10–19].

1. ОСНОВНЫЕ ПОЛОЖЕНИЯ ТЕОРИИ СТРУКТУРНОЙ ОРГАНИЗАЦИИ И ТИПОВ ДАННЫХ ЯП

В данной теории тип — математическое понятие, обозначающее множество значений элементов. Базовый тип в программировании — элементарный тип переменной (целое, вещественное и другие) в описании компонента, используемый внутри него. Тип присваивается переменной, определяющей класс значений, каждое из которых принадлежит одному и только одному типу. Операции над значениями типа — аксиомы, а операции над типом данных — функции установления взаимно однозначного соответствия передаваемых данных и необходимого преобразования значений одного типа данных в значения другого типа при их несовпадении [3–8, 20].

1.1. Синтаксические аспекты аксиоматической системы типов данных

Система включает фундаментальные типы данных (простые, структурные и сложные), множество операций и значений типов данных, их свойства и связи с другими типами данных. Простые типы — перечислимые типы данных, структурные — массивы и записи, сложные — множества, списки, последовательности и др. [3–8, 20].

Типы данных используются при описании функций, компонентов и программ в ЯП. Они реализуются системами программирования на разных платформах компьютеров как выходной код, который не только предназначен для выполнения программы, но и служит источником обеспечения взаимодействия в разнообразных, отличающихся между собой средах.

Аксиоматика простых типов данных. К простым типам данных относятся перечислимые типы, «целый — integer (i)», «вещественный — real (r)», «булевый — boolean (b)» и «символьный — character (c)». Они имеют вид

$$\text{type}\, T = X(x_1, x_2, \ldots . x_n),$$

где T — имя типа; $(x_1, x_2, \ldots, x_n)$ — имена значений из множества значений X.

Операции над перечислимыми типами множества Ω включают бинарные операции сопоставления и унарные операции pred и succ, задающие соответственно предыдущий и последующий элементы во множестве X. Операции сопоставления $(<, \le, >, \ge, =, \ne)$, далее $(\le)$, определяют линейный порядок элементов множества X.

Булевый тип определен на множестве значений X^b и операций Ω^b:

$$X^b = \{\text{false}, \text{true}\},\ \Omega^b = \{\&, \text{V}, \neg, \text{pred}, \text{succ}, \le\}.$$

Символьный тип определен на множествах значений X^c и операций Ω^c:

$$X^c = \{\ldots, 'A' \ldots, 'X' \ldots, '0', '1', \ldots, '9'\},\ \Omega^c = \{\text{pred}, \text{succ}, \text{ord}, \text{char}, \le\}.$$

Операция ord задает каждому символу его порядковый номер, а char — значение.

Аксиомы данных типов имеют вид

$$X.\min \in X,\ X.\max \in X,$$

$$(\forall x \in X) \& (x \ne X.\max) \Rightarrow \text{succ}(x) \in X,$$

$$(\forall x \in X) \& (x \ne X.\max) \Rightarrow \text{succ}(x) \ne X.\min,$$

где $X.\min$ и $X.\max$ — минимальный и максимальный элементы множества X.

Числовые типы (integer и real) имеют ограничения, связанные с архитектурой или явным описанием параметров компонентов. Им соответствуют базовые типы как отрезки вида

$$\text{type}\, T = (X.\min, \ldots, X.\max),$$

где $X.\min$ и $X.\max$ — минимальный и максимальный элементы отрезка. Для любого $x \in X$ выполняется условие $x.\min < x < x.\max$. Значение x зависит от реализации этих типов конкретным транслятором с учетом архитектуры компьютера.

Над переменными *целого* типа выполняются аналогичные операции, а также операции целочисленной арифметики: унарный минус, $+, -, \times, \text{div}$ и mod. Целый тип на отрезке определяется следующими аксиомами:

$$X^i = \{X^i.\min, X^i.\max + 1, \ldots X^i.\max\},$$

$$\Omega^i = \{+, \times, \text{div}, -, \le\},$$

$$\text{type}\, T^i = (X^i.\min, \ldots, X^i.\max).$$

Вещественный тип определяется с помощью операций сопоставления и обычных арифметических операций (унарный минус, $+, -, \times, /$). Аксиомы вещественного типа на отрезке имеют следующий вид:

$$X^r = \{x \mid X^r.\min \le x \le X^r.\max\},$$

$$\Omega^r = \{+, \times, /, -, \le\},$$

$$\text{type}\, T^r = (X^r.\min, \ldots, X^r.\max).$$

Любые типы данных приводятся к базовому типу, преобразуются к необходимому виду, заданному в программе, и после получения в ней результата базовый тип приводится к исходному с помощью следующих аксиом:

$$(\forall x \in X) \Rightarrow T(T^0(x)) = x,$$

$$(\forall x_1 \in X) \& (\forall x_2 \in X) \Rightarrow (x_1 \le x_2) \equiv (T^0(x_1) \le T^0(x_2)).$$

Здесь T^0 обозначает базовый тип для типа T. Операции преобразования значения $T^0(x)$ к $T(x)$ определяют соответствующий базовый тип при выполнении арифметических операций $\oplus$:

$$(\forall x_1 \in X) \& (\forall x_2 \in X) \Rightarrow (x_1 \oplus x_2) \equiv T(T^0(x_1) \oplus T^0(x_2)).$$

Аксиоматика структурных и сложных типов данных. К структурным типам данных относятся массив и запись.

Массив конструируется из базовых перечислимых типов и представляется механизмом отображения множества индексов I массива на множество значений Y элементов массива [3]: $M : I \to Y$. Над данным типом могут выполняться операции: упорядочение элементов массивов; сложение и вычитание элементов однотипных массивов; умножение двумерных массивов по правилам умножения матриц и др.

Операция умножения накладывает ограничения на область значений индексов массивов, определяемых математическими правилами умножения матриц. Операции сложения и вычитания выполняются для числовых массивов, входят в состав множества общих операций над массивами и имеют вид

$$X^\alpha = \{x \mid (\forall x_1 \in X^\alpha) \& (\forall x_2 \in X^\alpha) \Rightarrow I(x_1) =$$

$$= I(x_2) \& (Y(x_1) \cup Y(x_2) \subset \overline{Y}(X^\alpha))\},\ \Omega^\alpha = \{\le\},$$

$$\text{type}\, T^a = \text{array}\, T(I)\, \text{of}\, T(\overline{Y}),$$

где $I(x)$ — множество индексов x для массива; $Y(x)$ — множество значений элементов массива; $\overline{Y}(X^\alpha)$ — множество значений элементов для любых типов мас-

сива; T^a — тип данных *массив*; $T(I)$ — тип данных индексов массива; $T(\bar{Y})$ — тип данных для множества значений элементов массивов типа T^a.

К этому типу относятся только те массивы, у которых множества индексов совпадают, а множества значений их элементов принадлежат одному и тому же числовому множеству, характеризующему данный тип. Операции выполняются над массивами как над единым значением. Данные типа $T(Y)$ в описании массива T^a, в свою очередь, могут определяться рекурсивно через массив по следующей схеме:

$$\text{type}\, T^a = \text{array}\, T(I^1)\, \text{of}\, T(Y^1),$$
$$\text{type}\, T(Y^1) = \text{array}\, T(I^2)\, \text{of}\, T(Y^2),$$

что эквивалентно записи

$$\text{type}\, T^a = \text{array}\, T(I^1 \times I^2)\, \text{of}\, T(Y^2).$$

Множество индексов массивов, принадлежащих типу T^a, представлено в виде прямого произведения множеств значений для типов $T(I^1)$ и $T(I^2)$.

Запись представляет собой конкатенацию отдельных компонентов, которые могут иметь разные типы. Множество ее значений — прямое произведение значений ее компонентов. К множеству операций над записями относятся операции сопоставления однотипных структур (типы компонентов сравниваемых записей и порядок их следования одинаковы):

$$X^z = \{x \mid (x = x^{v1} \times \ldots \times x^{vn}) \& (x^{v1} \in X^{v1}) \& \ldots \& (x^{vn} \in X^{vn})\},\ \Omega^z = \{\leq\}.$$

Операции выполняются над записью как над единым структурным значением. К отдельным компонентам записи применяется операция селектора.

К сложным типам данных относятся: множество, объединение, список, последовательность, деревья и др. Некоторые из этих типов — стандартные в конкретных ЯП, другие реализуются через моделирование структур и операций над ними.

Множество. Общая форма представления имеет вид

$$\text{type}\, T = \text{powerset}\, T^0,$$

где T — тип множества; T^0 — базовый тип его элементов.

Тип T включает операции над множествами как над математическими типами — объединение, пересечение, разность, включение, тождественность и др. С помощью операций селектора осуществляется выбор типа T^0 из объекта типа T, а за счет операции конструирования — формирование из одного или из нескольких элементов типа T^0 объекта типа T.

Объединение. Общая форма представления имеет вид

$$\text{type}\, T = \text{union}\, (T^{v_1}, \ldots, T^{v_n}),$$

где T — тип объединения; $T^{v_1}, \ldots, T^{v_n}$ — базовые типы.

Любой объект типа T имеет значение и признак, по которому определяется один из типов $T^{v_1}, \ldots, T^{v_n}$ для данного значения. Механизм реализации объединения подобен механизму реализации вариантных записей.

Список. Конструктивный элемент ЯП Лисп, описывается как запись с одним или несколькими компонентами ссылочного типа, над которым выполняются соответствующие операции с записями, а также операции, применяемые к целому списку: выбор начального элемента, получение остатка списка, соединение и сравнение, инвертирование, поиск элементов и др.

Последовательность. Общая форма представления имеет вид

$$\text{type}\, T = \text{sequence}\, T^0,$$

где T — тип последовательности; T^0 — базовый тип.

Последовательность — один из вариантов типа списка, у которого каждый элемент содержит только одну ссылку для обеспечения односторонней связи. Операции над последовательностями аналогичны операциям над списками. Разновидность последовательности — строка, для которой каждый элемент содержит элемент символьного типа.

Деревья. Эти структуры используются для представления графов или других подобных объектов. Множество операций над деревьями аналогично множеству операций над списками, их реализация зависит от конкретных видов приложений.

Кроме описанных типов данных, в программировании используются таблицы, файлы и всевозможные комбинации указанных типов. Далее рассматривается семантика фундаментальных типов данных в компонентах на ЯП.

1.2. Семантические аспекты интероперабельности разнородных программ

С семантической точки зрения интероперабельность — обеспечение совместимости типов данных путем преобразования форм представления значений типов данных в программах ЯП к релевантному программному коду среды. Такое преобразование сводится к установлению взаимно однозначного соответствия (изоморфизма) значений типов данных вызывающего и вызываемого компонентов в среде выполнения и включает следующие случаи их изоморфного отображения.

1. Изоморфизм существует между множествами значений типов данных ЯП и множеством операций.

2. Изоморфизм существует при некоторых ограничениях между множествами значений типов данных ЯП и разными операциями.

3. Изоморфизм не существует при отображении множества значений типов данных разных ЯП.

Изоморфизм отображений множества значений t типов данных T с помощью операций Ω определяется мощностью системы преобразования или мощностью множества значений типов X_{α}^{t} [9]. Это положение — критерий оценки правильности взаимодействия программ для случая, если одинаковые типы данных в разных ЯП имеют различные множества значений или если осуществляется адаптация разноязыковых компонентов с одного типа компьютера для другого (например, диапазоны целых и вещественных чисел различны). Любые два элемента сравнимы, если изоморфное отображение сохраняет отношение линейного порядка.

Лемма. Для любого изоморфного отображения φ между системами отображений выполняются равенства

$$\varphi(X_{\alpha.\min}^{t}) = X_{\beta.\min}^{q},\ \varphi(X_{\alpha.\max}^{t}) = X_{\beta.\max}^{q}.$$

Доказательство данной леммы простое. Для всех построенных систем преобразования, описывающих простые типы данных, множества значений ограничены. Структурные типы данных строятся из простых с помощью конечного числа операций, а их множества значений также конечны. Поэтому $X_{\alpha.\min}^{t}$, $X_{\alpha.\max}^{t}$, $X_{\beta.\min}^{q}$, $X_{\beta.\max}^{q}$ существуют и принадлежат множествам X_{α}^{t} и X_{β}^{q} соответственно. С учетом линейной упорядоченности этих множеств выполняется условие леммы. В противном случае изоморфное отображение не сохраняло бы линейный порядок, что противоречит сделанному ранее выводу. Лемма доказана.

Системы изоморфного отображения между базовыми типами данных тривиальны. Для более сложных структур данных преобразование сводится к простым типам данных. При выполнении операций преобразования массивов и записей возможны комбинации: массив — в массив, запись — в запись, массив — в запись, запись — в массив. Эти преобразования сохраняют линейный порядок.

Рассмотрим изоморфизм типа данных «массив — в массив». В нем взаимно однозначное отображение φ сохраняет линейный порядок. В выражении $(\forall x^a_{\alpha_1} \in X^a_\alpha)\,\&$ $\&\,(\forall x^a_{\alpha_2} \in X^a_\alpha)\,\&\,(x^a_{\alpha_1} \le x^a_{\alpha_2}) \Rightarrow \varphi(x^a_{\alpha_1}) \le \varphi(x^a_{\alpha_2})$ отношение $\le$ означает его выполнение для всех элементов области определения. Функции $\varphi(x^a_{\alpha_1})$ и $\varphi(x^a_{\alpha_2})$ обеспечивают преобразование индексов i и значений y для элементов массива: $\varphi_i : I^a_\alpha \to I^a_\beta$, $\varphi_y : \overline{Y}(X^a_\alpha) \to \overline{Y}(X^a_\beta)$.

Функции $\varphi(x^a_{\alpha_1})$ и $\varphi(x^a_{\alpha_2})$ будут определяться как ограничения отображения φ_y на соответствующем подмножестве и обозначаться $\varphi_y | Y(x^a_{\alpha_1})$ и $\varphi_y | Y(x^a_{\alpha_2})$. Неравенство $\varphi(x^a_{\alpha_1}) \le \varphi(x^a_{\alpha_2})$ будет эквивалентным $\varphi_y | Y(x^a_{\alpha_1}) \le \varphi_y | Y(x^a_{\alpha_2})$.

Аналогичную концепцию преобразования имеет «запись – в запись». Преобразование «запись – в массив» требует, чтобы типы всех элементов записи были одинаковыми, а сама запись представлялась в виде массива. Множество значений элементов записи образует множество значений элементов массива Y. Множество индексов переупорядочивает компоненты записи: i-й компонент ставится в соответствие $\varphi(i)$-му элементу во множестве J. Множество I является множеством значений перечислимого типа.

Рассмотренные типы данных используются в программах и компонентах в ЯП. Обеспечение взаимодействия компонентов в разных средах осуществляется с помощью их интерфейсов, которые описываются языками MIL, IDL, API, RMI и др. В этих языках интерфейс является специальным разделом описания параметров, передаваемых другим компонентам. Далее рассматриваются теоретические и прикладные аспекты создания ПС из компонентов с возможностью их взаимодействия при выполнении в современных средах.

2. ФОРМАЛИЗАЦИЯ ИНТЕРФЕЙСА КОМПОНЕНТОВ

Основной особенностью современного программирования является использование в новых системах готовых компонентов (reuse), которых в информационном пространстве разработано большое количество. Процесс объединения отдельных компонентов, компонентов повторного использования (КПИ), каркасов, контейнеров в более сложную компонентную структуру задается описанием связей (ассоциаций) и утверждениями об их взаимодействии. Сборку компонентов можно формализовать теориями первого порядка, сложный каркас — предикатами описания классов, состояний переменных, методов и их отношений. Класс и объект задаются сортами объектов первого класса, при этом могут использоваться сорта bool и int.

КПИ включает некоторый метод (или их совокупность) с определенной сигнатурой и типами данных, которые передаются другому КПИ и возвращаются после реализации этого метода. Повторное использование компонентов (reusability) представляет собой систематическую деятельность как в плане классификации и их каталогизации в хранилищах (репозитариях), так и применения в новых разработках программных и информационных систем [10–19].

По своей природе программный компонент является независимым от ЯП самостоятельно реализованным объектом, обеспечивающим выполнение определенной совокупности прикладных сервисов, доступ к которым возможен через интерфейсы, указывающие функции и операции обращения к компоненту. При этом интерфейс содержит описание характеристик и атрибутов компонента, который обменивается данными с другими компонентами в компонентной или иной среде.

Компонентная среда — расширение классической модели «клиент–сервер» с множеством серверов компонентов (или серверов приложений — application servers). Компонент, представленный как контейнер, разворачивается внутри сервера. Для каждого сервера может существовать произвольное количество контейне-

ров. Взаимосвязь и взаимодействие контейнера с сервером регламентированы стандартизированным интерфейсом. Контейнер управляет порождаемыми им экземплярами компонента — реализациями соответствующей функциональности.

С каждым контейнером связано два типа интерфейсов — интерфейс для взаимодействия с другими компонентами и интерфейс системных сервисов, необходимых для его функционирования и реализации специальных функций (например, поддержка распределенных транзакций с участием нескольких компонентов). Первый тип интерфейса (Home interface) обеспечивает управление экземплярами компонента с обязательными реализациями методов поиска, создания и удаления отдельных экземпляров. Ко второму типу относятся интерфейсы, которые обеспечивают доступ к реализации функциональности компонента. Фактически с каждым экземпляром связан определенный функциональный интерфейс.

Определим требования к интерфейсам.

1. Компонент содержит один или больше интерфейсов, которые имеют уникальные имена, и описание структуры интерфейса.

2. Каждый интерфейс состоит из нескольких операций (методов) или из одной. Каждая операция имеет имя, множество входных и выходных параметров, тип которых определяется на множестве простых и структурных типов данных ЯП. Совокупность параметров и их типов — сигнатура операции, а совокупность сигнатур всех операций — сигнатура интерфейса.

3. Каждый компонент реализует схему взаимодействия — сервер и клиент. Если к компоненту обращаются другие компоненты, то это — сервер. Для поддержки определенного сервиса компонент обращается к клиенту. Интерфейсы делятся на два типа. К первому типу относятся интерфейсы, которые непосредственно описывают сервисные возможности (функциональность) компонента, ко второму — ссылки на интерфейсы других компонентов, операции которых необходимо выполнить для функционирования компонента.

4. Взаимодействие между компонентами определяется интерфейсами. Объекты, между которыми может быть скрытое (hidden) взаимодействие, входят в состав одного компонента и рассматриваются как единое целое. Интерфейсы компонентов открыты, задают их внешнее поведение, в середине они недоступны.

2.1. Типы отношений между компонентами

Наследование. Данное отношение двух компонентов характеризует отношение наследования входных интерфейсов.

Экземпляризация. Выражение $CIns_k^{ij} = (IIns_k^{ij}, IntFunc^i, ImpFunc^j)$ описывает определенный экземпляр компонента *Comp*, где $IIns_k^{ij}$ — уникальный идентификатор экземпляра; $IntFunc^i$ — функциональность интерфейса $CIntI^i \in CInt$; $ImpFunc^j$ — программный элемент, обеспечивающий выполнение реализации $CImp^i \in CImp$.

Контракт. Между компонентами $Comp_1$ и $Comp_2$ существует отношение контракта $Cont_{12}^{im} = (CInt_1^u, CInt_2^m, IMap_{12}^{im})$, где $CInt_1^u \in CInt_1$ — исходный интерфейс первого компонента, $CInt_2^m \in CInt_2$ — входной интерфейс второго; $IMap_{12}^{im}$ — отображение соответствия между методами обоих интерфейсов с учетом сигнатур и типов данных, которые передаются, если компонент $Comp_2$ имеет реализацию для интерфейса $CInt_2^m$, обеспечивающую выполнение функциональности $IntFunc_1^u$ интерфейса $CInt_1^u$.

Связывание. Если между компонентами $Comp_1$ и $Comp_2$ существует отношение контракта $Cont_{12}^{im}$, то между их экземплярами $CIns_{1k}^{ij} = (IIns_{1k}^{ij}, IntFunc^i, ImpFunc^j)$ и $CIns_{2p}^{mq} = (IIns_{2p}^{mq}, IntFunc^m, ImpFunc^q)$ существует отношение связывания относительно контракта $Cont_{12}^{im}$ в виде $Bind(IIns_{1k}^{ij}, IIns_{2p}^{mq}, Cont_{12}^{im})$.

К формальным основам компонентного программирования относятся разработанные формальные модели, компонентная алгебра с операциями изменения компонентов и интерфейсов, а также подход к реализации интерфейса компонентов [9, 20–23].

2.2. Модели компонентного программирования

Модели компонентного программирования обеспечивают переход от теории к практике (например, для таких сред, как CORBA, COM, EJB). Ряд моделей, разработанных на единой понятийной, терминологической и математической основе, предложены в [9, 10]. Они включают: модель компонента; модель интерфейса; модель компонентной среды; компонентную алгебру.

Модель компонента имеет вид

$$Comp = (CName, CInt, CFact, CImp, CServ), \tag{1}$$

где $CName$ — уникальное имя компонента; $CInt = \{CInt^i\}$ — множество интерфейсов, связанных с компонентом; $CFact$ — интерфейс обеспечения функций управления экземплярами компонента; $CImp = \{CImp^j\}$ — множество реализаций компонента; $CServ = \{CServ^r\}$ — множество общесистемных сервисов.

Множество $CInt = CInt^1 \cup CInt^2$ состоит из входных $CInt^1$ и выходных $CInt^2$ интерфейсов. Отличие между ними заключается в том, что для выходных интерфейсов компонент имеет собственные реализации, а для входных реализации находятся в других компонентах. Каждый $CInt^i \in CInt$ имеет модель

$$CInt^i = (IntName^i, IntFunc^i, IntSpec^i),$$

где $IntName^i$ — имя интерфейса; $IntFunc^i$ — функциональность (совокупность методов); $IntSpec^i$ — спецификация интерфейса: описание типов, констант, других элементов, сигнатур методов и т.п.

Интерфейс $CFact$ определяет методы, необходимые для управления экземплярами компонента (поиск, создание, уничтожение и т.п.).

Каждая реализация $CImp^j \in CImp$ имеет модель

$$CImp^j = (ImpName^j, ImpFunc^j, ImpSpec^j),$$

где $ImpName^j$ — идентификатор реализации; $ImpFunc^j$ — программный элемент, обеспечивающий выполнение функциональности для реализации; $ImpSpec^j$ — спецификация реализации (условия выполнения, зависимости от определенных платформ и т.п.).

Необходимым требованием существования компонента является условие его целостности

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

где $Provide(CInt^i)$ — функциональность, обеспечивающая реализацию методов интерфейса $CInt^i$. Для взаимодействия компонентов $Comp_1$ и $Comp_2$ необходимо выполнение следующего условия: если $CInt_1^i \in CInt_1$, то должен существовать $CInt_2^k \in CInt_2$ такой, что

$$Sign(CInt_1^i) = Sign(CInt_2^k) \& Provide(CInt_1^{i)} \subseteq CImp_2^j,$$

где $Sign(\ldots)$ — сигнатура соответствующего интерфейса.

Модель интерфейса компонента имеет вид

$$CInt^i = (IntName^i, IntFunc^i, IntSpec^i),$$

где $IntName^i$ — имя i-го 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

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

где $Provide(CInt^i)$ — функциональность, обеспечивающая реализацию методов интерфейса $CInt^i$.

Наличие знака включения в данной формуле означает, что выбранная реализация компонента может обеспечить поддержку не только нужного интерфейса, но и других. Например, практические технологии и ЯП (CORBA, Java, C++ и другие) содержат необходимые средства. Для каждого из интерфейсов может существовать несколько реализаций, которые различаются особенностями функционирования (например, операционной средой, средствами сохранения данных и т.д.).

Взаимодействие компонентов $Comp_1$ и $Comp_2$ определяется условием:

если $CInt_1^i \in CInt_1$, то должен существовать $CInt_2^k \in CInt_2$ такой, что

$$Sign(CInt_1^i) = Sign(CInt_2^k) \,\&\, Provide(CInt_1^i) \subseteq CImp_2^j,$$

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

Рассмотрим свойства компонентов.

1. Два компонента, $Comp_1$ и $Comp_2$, тождественны (равны), если тождественны их соответствующие элементы. Как следствие, замена $Comp_1$ на $Comp_2$ не влияет на компонентную программу, к которой принадлежит $Comp_1$.

2. Два компонента, $Comp_1$ и $Comp_2$, эквивалентны, если тождественны их множества интерфейсов и реализаций. Замена $Comp_1$ на $Comp_2$ не изменяет функциональности компонентной программы при условии соответствия имен в самой программе.

3. Два компонента, $Comp_1$ и $Comp_2$, подобны, если тождественны их множества интерфейсов. Замена $Comp_1$ на $Comp_2$ сохраняет взаимосвязи компонентов, функциональность компонентной программы может измениться.

Модель компонентной среды имеет вид

$$CE = (CName, IntRep, ImpRep, CServ, CServImp), \qquad (2)$$

где $CName = \{CName^m\}$ — множество имен компонентов, которые входят в состав среды; $IntRep = \{IntRep^i\}$ — репозитарий интерфейсов компонентов среды; $ImpRep = \{ImpRep^j\}$ — репозитарий реализаций; $CServ = \{CServ^r\}$ — интерфейс системных сервисов; $CServImp = \{CServImp^r\}$ — множество реализаций системных сервисов.

Каждый элемент $IntRep$ задается парой $(CInt^i, CName^m)$, где $CInt^i$ — интерфейс компонента, а $CName^m$ — имя компонента, реализующего интерфейс. Аналогично задается элемент $ImpRep$ $(CImp^j, CName^m)$, где $CImp^j$ — реализация компонента.

Компонентная среда — множество серверов приложений, где разворачиваются контейнеры, экземпляры которых — реализация функциональности компонента. Взаимосвязь контейнера с сервером обеспечивается через стандартизированные интерфейсы $CFact$. Связь между компонентами, развернутыми в разных серверах, обеспечивается реализациями интерфейса $CServ$.

Каркас компонентной среды определяется тем, что $CName$, $IntRep$, $ImpRep$ — пустые множества, т.е.

$$FW = (\varnothing, \varnothing, \varnothing, CServ, CServImp).$$

Пусть $FW_1 = (\varnothing, \varnothing, \varnothing, CServ_1, CServImp_1)$ и $FW_2 = (\varnothing, \varnothing, \varnothing, CServ_2, CServImp_2)$ — два каркаса.

Каркас FW_1 совместим с каркасом FW_2, если существует отображение $SMap: CServ_1 \to CServ_2$ такое, что $SMap(CServ_1) \subseteq CServ_2$.

Компонентная алгебра содержит операции внешнего и внутреннего типа.

Внешняя компонентная алгебра определяет множество операций над компонентами и компонентными средами и задана выражением

$$\Psi = \{CSet, CESet, \Omega\}, \tag{3}$$

где *CSet* — множество компонентов, каждый из которых описывается выражением (1); *CESet* — множество компонентных сред, каждая из которых описывается выражением (2); Ω — множество операций типа

- инсталляция (развертывание компонента): $CE_2 = Comp \oplus CE_1$;
- объединение компонентных сред: $CE_3 = CE_1 \cup CE_2$;
- удаление компонента из компонентной среды: $CE_2 = CE_1 \setminus Comp$.

Теорема 1. Каждая компонентная среда *CE* — результат выполнения операций развертывания компонентов из состава компонентного каркаса:

$$CE = Comp_1 \oplus Comp_2 \oplus \ldots \oplus Comp_n \oplus FW.$$

Теорема 2. Построение компонентной среды не зависит от порядка инсталляции компонентов, которые входят в ее состав, т.е.

$$Comp_1 \oplus (Comp_2 \oplus CE) = Comp_2 \oplus (Comp_1 \oplus CE).$$

Теорема 3. Операция объединения компонентных сред ассоциативна и коммутативна:

$$(CE_1 \cup CE_2) \cup CE_3 = CE_1 \cup (CE_2 \cup CE_3),$$

$$CE_1 \cup CE_2 = CE_2 \cup CE_1.$$

Теорема 4. Для любой компонентной среды $CE \cup FW = FW \cup CE = CE$.

Теорема 5. Для произвольных компонентных сред CE_1, CE_2 и компонента *Comp* всегда выполняется

$$Comp \oplus (CE_1 \cup CE_2) = (Comp \oplus CE_1) \cup CE_2 = (Comp \oplus CE_2) \cup CE_1.$$

Теорема 6. Для любого компонента *Comp* и компонентной среды *CE* всегда выполняется

$$(Comp \oplus CE) \setminus Comp = CE.$$

Модель компонентной программы имеет вид $CP = (CE, Cont, Comp)$. *Comp* — подмножество компонентов *CE*, включающих реализации, для которых отсутствуют выходные интерфейсы, и обращающихся к другим компонентам с помощью входных интерфейсов.

Если для каждого компонента $Comp_1$ в *CE* имеется входной интерфейс $CInt_1^u$, компонент $Comp_2$ с соответствующим интерфейсом $CInt_2^m$ и контракт $Cont_{12}^{im} = (CInt_1^u, CInt_2^m, IMap_{12}^{im})$ в составе множества *Cont*, то это является условием целостности компонентной программы.

Внутренняя компонентная алгебра содержит операции, обеспечивающие различного вида преобразования внутри компонентов независимо от среды. Они позволяют осуществлять изменения компонентов и их интерфейсов.

Внутренняя алгебра $\Xi = \{\Xi_1, \Xi_2, \Xi_3\}$ включает:

$\Xi_1 = \{CSet, O_{Refac}\}$ — подалгебру из множества компонентов *CSet* и операций O_{Refac} рефакторинга над ними;

$\Xi_2 = \{CSet, O_{Reing}\}$ — подалгебру из множества компонентов *Cset* и операций O_{Reing} реинженерии отдельных компонентов;

$\Xi_3 = \{CSet, O_{Rever}\}$ — подалгебру из множества компонентов *CSet* и операций O_{Rever} реверсной инженерии.

К операциям данной алгебры относятся операции изменения, добавления интерфейсов или реализации компонентов, а также перепрограммирования и реструктуризации при изменении, обновлении их функциональности в целях повторного применения.

Множество операций **рефакторинга**:

$$O_{Refac} = \{AddOImp, ReplImp, AddInt, Cfact\},$$

где *AddOImp* — операция добавления операции для существующего интерфейса и нового; *ReplImp* — операция замещения существующей реализации новой; *AddInt* — операция добавления нового входного интерфейса; *Cfact* — операция расширения интерфейса.

Операция *AddOImp* добавляет для существующего интерфейса новый интерфейс: $NewComp = AddOImp(OldComp, NewCImp^s, NewCIntO^s)$.

Условие целостности компонента выполняется автоматически, так как множество входных интерфейсов остается прежним и из целостности исходного компонента вытекает целостность нового. Операция *AddOImp* ассоциативна и коммутативна.

Операция *ReplImp* замещает существующую реализацию новой реализацией: $NewComp = ReplImp(OldComp, NewCImp^s, NewCIntO^s, OldCImp^r, OldCIntO^r)$. В ней $NewCImp^s$ — новая реализация; $NewCIntO^s$ — множество дополнительных исходных интерфейсов для добавленной реализации; $OldCImp^r$ — замещаемая реализация; $OldCIntO^r$ — множество исходных интерфейсов, связанных с замещаемой реализацией.

Множество операций **реинженерии**:

$$O_{Reing} = \{rewrite, restruc, adop, supp, conver\},$$

где *rewrite* — операция перепрограммирования текста компонента; *restruc* — операция реструктуризации компонента; *adop* — операция адаптации компонента к новым условиям; *supp* — операция добавления новых функций; *conver* — операция конвертирования данных компонентов.

Эти операции изменяют только компонент, без изменения интерфейса, поскольку предполагается, что в качестве интерфейса используется запрос (request) к компоненту в среде, где он располагается и выполняется. Запрос формируется в среде клиента и отправляется через сеть серверу для выполнения в нем вызванного компонента.

Пусть $OldComp = (OldCName, OldCImp)$ обозначает унаследованную реализацию компонента, а $NewComp = (NewCName, NewCImp)$ — реализацию после выполнения операции реинженерии $(OldCInt, CFact, OldCImp, Cserv = \varnothing)$. В результате выполнения операций реинженерии получается набор новых компонентов:

$$NewComp^1 = rewrite(OldComp, NewCImp),$$
$$NewComp^2 = restruc(OldComp, NewCImp),$$
$$NewComp^3 = adop(OldComp, NewCImp),$$
$$NewComp^4 = supp(OldComp, NewCImp),$$
$$NewComp^5 = conver(OldComp, NewCImp).$$

Они образуют множество компонентов $Cset = \{NewComp^n\}$, где $n = 1\ldots5$. Все операции реинженерии ассоциативны, коммутативны и обеспечивают целостность нового компонента.

Множество операций **реверсной инженерии**:

$$O_{Rever} = \{restruc, design, rewrite\},$$

где *restruc* — операция реструктуризации системы; *design* — операция построения нового компонента системы из устаревшей, удовлетворяющего новым условиям; *rewrite* — операция перевода компонента на другой язык программирования соответственно новой структуре системы.

Рассмотренный метод реверсной инженерии дает возможность решать задачи эволюционного развития путем преобразования кодов устаревших систем.

Алгебра Ψ компонентного программирования включает внешнюю и внутреннюю алгебры и имеет общий вид

$$\Psi = \{CSet, CESet, \Omega\} \cap \Xi = \{CSet, CESet, \Xi_1, \Xi_2, \Xi_3\}.$$

2.3. Подход к реализации интерфейса компонентов

В процессе построения компонентной программы используются операции компонентной алгебры, развертывания компонентов, создания компонентной среды, определения исходных компонентов, формирования множества контрактов и др.

Функциональность компонентов задается средствами разных ЯП с соответствующими в них множествами типов данных. Интеграция разноязыковых компонентов осуществляется с помощью разработанных формальных механизмов релевантного преобразования несовпадающих типов данных, которые передаются при взаимодействии компонентов в интегрированной распределенной среде.

Исторически сформировались такие подходы к ее реализации: 1) преобразование типов данных, передаваемых от клиента к серверу и обратно, осуществляется с помощью модуля посредника типа stub (например, в системе CORBA); 2) при каждом вызове компонента предварительно подготавливаются необходимые значения передаваемых типов данных в соответствующем формате вызываемого компонента и наоборот; 3) системы программирования с ЯП разных компонентов генерируют совместимый промежуточный или выходной код [9].

В теоретическом плане принципиального различия между этими подходами не существует, так как главным условием интеграции двух разноязыковых компонентов является наличие между соответствующими ЯП описания компонентов формальных отображений, представленных как суперпозиции базовых отображений.

Формально модель интеграции разноязыковых компонентов, заданных на множестве компонентов $CSet = \{Comp_i\}$, обеспечивает взаимодействие компонентов, обменивающихся данными, каждое из которых определяется тройкой: именем переменной, ее типом данных и значением. Обмениваемые данные каждой пары компонентов $Comp_i$ и $Comp_j$ могут быть эквивалентными, если они имеют одинаковую семантическую структуру и типы данных, или неэквивалентными, тогда необходимо их преобразование с помощью функций отображений

$$FN_{ij} : N_u \to N_j,\ FT_{ij} : T_i \to T_j,\ FV_{ij} : V_i \to V_j,$$

где FN_{ij} устанавливают соответствие между именами переменных (например, во множествах формальных и фактических параметров); FT_{ij} описывают эквивалентные отображения для типов данных; FV_{ij} реализуют необходимые преобразования значений данных для неэквивалентных типов данных.

Задача построения преобразования FN_{ij} решается путем установления соответствия имен переменных (например, в описании конфигурации для компонентов, которые интегрируются). Отображение между типами данных FT_{ij} базируется на преобразованиях типов данных, каждый из которых подается как абстрактная алгебра или алгебраическая система $T = (X, \Omega)$, где X — множество значений, которые могут принимать значения этого типа, а Ω — множество операций над этими переменными. Отображения FV_{ij} применяются при неэквивалентности типов T_i и T_j (например, преобразование целых значений в действительные).

В случаях многоразовых вызовов компонентов необходимо выполнение прямого и обратного преобразования между формальными и фактическими параметрами, при этом важное условие состоит в том, чтобы отображения между соответствующими типами были изоморфны, т.е. существовали изоморфизмы между алгебраическими системами, которые описывают эти типы данных.

Под преобразованием типа $T_i = (X_i, \Omega_u)$ в тип $T_j = (X_j, \Omega_j)$ понимается такое преобразование, при котором семантическое содержание операций из Ω_u эквивалент-

но содержанию операции из Ω_j. В общем случае преобразование типа T_i в тип T_j может быть односторонним, т.е. эквивалентные преобразования не существуют.

Задача обеспечения взаимосвязи пары компонентов, разработка которых выполнена на разных ЯП, состоит в построении совокупности отображений для всех вызовов методов, каждое из которых устанавливает однозначное соответствие между множеством фактических параметров $V = \{v_1, v_2, \ldots, v_k\}$ для вызываемого компонента и множеством формальных параметров $F = \{f_1, f_2, \ldots, f_l\}$ для вызывающего компонента.

Для основных типов данных ЯП, применяемых для описания модулей или компонентов, в работах [3, 4] определены алгебраические системы, построены изоморфные отображения между этими системами и сформированы условия существования таких отображений.

Рассмотренные формальные средства являются общими и могут адаптироваться к принципам взаимодействия компонентов в гетерогенных средах, например в среде Grid [14–16]. Задача устранения отличий в представлении данных на разных ЯП и компьютерных платформах гетерогенной среды может быть реализована новым методом, описываемым далее.

3. КОНЦЕПЦИЯ ГЕНЕРАЦИИ ОБЩИХ ТИПОВ ДАННЫХ КОМПОНЕНТОВ ДЛЯ ГЕТЕРОГЕННЫХ СРЕД

3.1. Подход к организации интероперабельных вычислений в гетерогенной среде Grid

С 2007 г. начал действовать европейский проект Grid — сетевая инфраструктура для организации распределенных вычислений в задачах по разным научным направлениям (физика, математика, медицина, биология и др.) [14–16]. Этот проект включает ряд подпроектов систем: GCube, ETICS и др. В частности, система ETICS предназначена для автоматизированного построения, конфигурирования, интеграции и тестирования разного рода программ и систем для решения научно-исследовательских задач. Ее архитектура базируется на каркасе FP6 EC, ориентированном на производство распределенных систем путем интеграции существующих процедур, инструментальных средств и ресурсов данной инфраструктуры с применением Веб-портала для управления мультиплатформенными ресурсами среды Grid.

ETICS содержит типовой набор характеристик и процедур для построения и тестирования новых пакетов и услуг. Он будет расширяться путем добавления новых плагинов (plugins) [14] со специализированными услугами для конкретной области знаний. Каждый плагин включает публичный интерфейс с описанием услуг для потребителей или поставщиков. Первоначально данный набор охватывает средства управления с рабочих мест заданиями, связанными с ОС, архитектурой CPU и компиляторами в ЯП. Кроме того, в него входит механизм спецификации зависимостей между различными пакетами и их тестами, которые автоматически управляют построением, тестированием, компилированием или развертыванием. Базовое множество функциональных плагинов предназначено для проверки договоров, тестов выполнения разных элементов систем, генерации документации и ведения готовых объектов программ в оперативном или постоянном репозитариях ETICS.

Технология создания больших наборов пакетов из исходных или комбинаций перекомпилированных двоичных элементов поддерживается процессом доступа к репозитариям для формирования распределенной версии и ее репродукции. Скомпилированная структура автоматически находит и загружает из репозитария подходящие двоичные элементы. При этом существенным препятствием здесь, как и в других рассмотренных средах, является проблема отображения элементов системы на альтернативную платформу. Она решается путем построения перекрестных ссылок (cross platform) к требуемой платформе и генерации модулей, расставляющих необходимые флажки в скомпилированном коде (например, при выполнении готовых 32-разрядных двоичных элементов на 64-разрядных платформах) сетевой среды Grid.

Следующая важная проблема — стандартизация описания типов данных для строящихся в системе ETICS трех главных объектов: Проект, Подсистема и Компонент. Проект может состоять из Подсистем или Компонентов. Подсистема может содержать только Компоненты. В системе предложена модель данных с типовым форматом CIM для взаимоотношений между различными объектами. Модель данных, как и модель CIM, позволяет вводить формальные сущности в структуру проектов ПС, описывать объекты и взаимоотношения между ними, а также предоставлять результаты выполнения ПС. Внутреннее хранение данных основывается на модели данных реляционного типа, реализованной средствами MySQL [19].

Описание модели данных основано на следующих положениях:

— каждый компонент содержит описание свойств (имя, лицензия, URL репозитария и т.д.) и глобального уникального идентификатора — ID (GUID);

— объект конфигурации содержит информацию о версии, связи с репозитарием, GUID, виде платформы и связи своего идентификатора с GUID компонента;

— объект платформы содержит команды проверки (checkout) скомпилированного компонента, тестовых команд и GUIDs, а также взаимоотношения связи с каждой конфигурацией;

— при определении конфигурации и платформы в каждом объекте объявляются GUID, его свойства, среда выполнения и зависимости, которые могут быть статическими или динамическими. Статическая зависимость — взаимоотношение между двумя конфигурациями, динамическая — между конфигурацией и компонентом.

3.2. Характеристика общих типов данных стандарта

Для обеспечения взаимодействия между разнородными программными объектами, реализованными в ЯП для современных и будущих сред, разработан стандарт ISO/IEC 11404:2007, предоставляющий формальный математический аппарат спецификации типов данных в общем виде, который независим от синтаксиса описания фундаментальных типов данных в ЯП. Этот стандарт содержит формализмы для описания примитивных, агрегатных и генерируемых типов данных, механизмы их агрегации и генерации, а также процедуры преобразования данных к внешнему ЯП и внутреннему языку стандарта и обратно.

Стандарт включает также формальные механизмы задания параметров интерфейса в языках (IDL, API, RPC) при вызове или обращении к процедурам, системам, компонентам повторного использования, относящимся к готовому сервису среды. Стандартный интерфейс базируется на языковых преобразованиях вида: «*<язык> обращение <сервис>*». Каждый интерфейс к сервису включает параметры, их типы данных, которые потребуется преобразовывать к стандартным типам данных и наоборот.

В стандарте описаны процедуры генерации типов данных к форме XML-документов. Все общие типы данных ориентированы на генерацию других типов данных, которые существуют в ЯП для сохранения преемственности. В разделе «declaration» дается синтаксис и семантика описания GDT — типов данных, принципы генерации новых типов данных и их агрегации. Каждый тип данных имеет шаблон описания, спецификатор типа данных, его значение в пространстве значений и операции над типами данных. Общий тип данных включает:

— концептуальное или абстрактное понятие, характеризующее его по значению и свойствам;

— структурное понятие, характеризующее этот тип данных как концептуальный, стандартный интерфейс к сервису;

— реализационное понятие, определяемое правилами представления типа данных в заданной среде.

К абстрактным нотациям относятся примитивные и не примитивные типы данных, базирующиеся на понятии структурной организации [4–7], применяемой в описании фундаментальных типов данных ЯП и интерфейсов программ. Структурные нотации используются для спецификации сложных типов данных и словарей реализации типов данных, отличающихся от концептуальных понятий спосо-

бом их идентификации. Преобразование общих типов данных в типы данных других ЯП или новых языков — основа реализационного понятия.

Новые системы обработки информации, которые строятся с учетом этого стандарта, базируются на принципах реализации типов данных, спецификации интерфейсов, программ обработки и обмена информацией, с использованием соответствующей системы нотаций для типов данных. Ввиду разнообразия типов данных и их нотаций системы должны обеспечивать описание и преобразование внутренних типов данных к типам данных стандарта или ЯП, а именно:

— описание типов данных, механизмов их генерации, а также нотации и значения из пространства значений, определенных стандартом;

— операции над общими типами данных, характеристические операции для организации работы с типами данных стандарта;

— набор типов данных стандарта, для которых обеспечиваются стандартные внутренние и внешние преобразования.

Данный стандарт будет поддерживаться другими стандартами, формализующими средства преобразования типов данных в стандарте языка к промежуточным (параметризованным) типам данных, в том числе и к ЯП [24]. Для этого в нем имеются механизмы перехода от параметризованного типа к общему. Параметрические значения определяются пользователями или разработчиком стандарта. Например, в стандарте языка определяется тип данных integer, а соответствующий процессор реализует некоторый диапазон этого типа — integer range (min ... max) и значения min и max.

Предложенные в стандарте рекомендации, а также средства описания типов данных и методов их преобразования являются общими. Новый вариант стандарта GDT дает общее описание типов данных, которые могут использоваться в действующих и вновь создаваемых ЯП, а также иметь для них программную поддержку.

3.3. Новый подход к организации взаимодействия компонентов на основе GDT

Реализация требований стандарта может быть осуществлена специальными средствами (рис. 1).

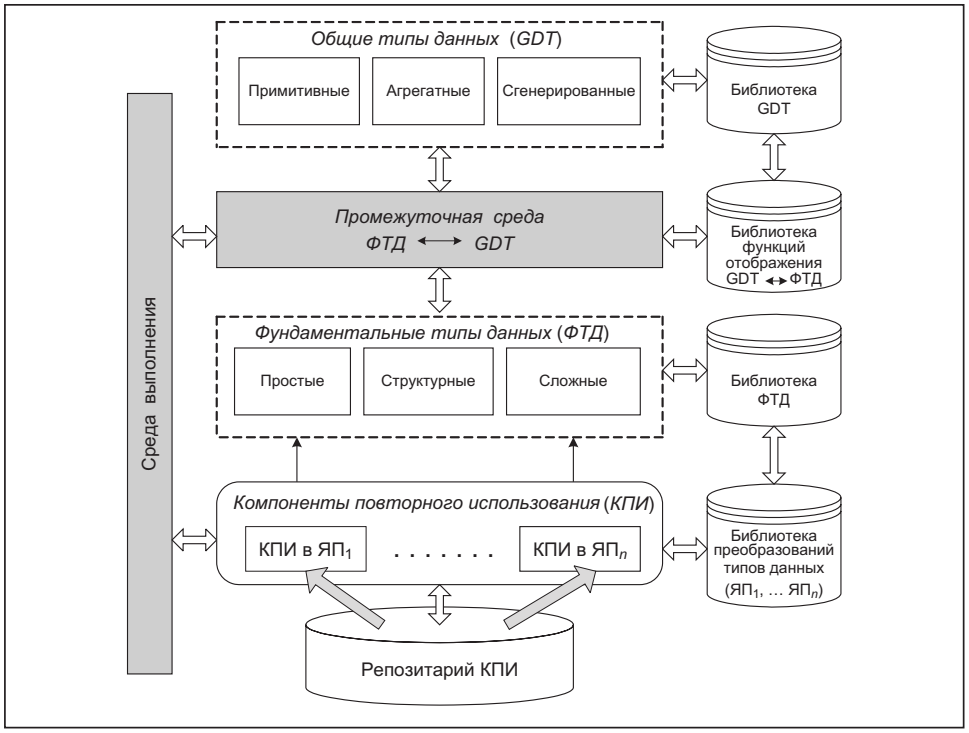

Рис. 1. Схема обработки типов данных GDT и ФТД

Суть этих средств заключается в следующем [25]:

1) спецификация внешних типов данных компонентов, подсистем и систем средствами языка GDT и их накопление в одном из репозитариев;

2) создание ряда библиотек функций для преобразования типов данных GDT (примитивных, агрегатных и сгенерированных) к ФТД (простым, структурным и сложным) языков программирования, с помощью которых будет генерироваться необходимая промежуточная среда взаимодействия разноязыковых компонентов, подсистем и проектов;

3) анализ и разработка системы управления автоматизированной генерацией для каждой взаимодействующей пары элементов проекта, нового элемента типа stub, содержащего обращение к соответствующим функциям преобразования типа данного к необходимому виду при передаче информации взаимодействующему компоненту и обратно, аналогично подходу к реализации интерфейса в CORBA [22], АПРОП [9].

Таким образом, постоянно возникающую проблему интероперабельности разнородных компонентов из-за появления новых ЯП, архитектур платформ и сред можно решить с помощью программного инструментария, основанного на GDT, адекватного по отношению к новым видам разнородных программных объектов.

ЗАКЛЮЧЕНИЕ

В работе определено понятие интероперабельности разнородных компонентов и систем и связанного с ним базиса описания фундаментальных типов данных ЯП разных поколений. Рассмотрена проблема взаимодействия компонентов с теоретической, семантической и реализационной точки зрения. Основу теории составляет структурная организации типов данных К. Хоара, исходя из которой представлена аксиоматика фундаментальных простых, структурных и сложных типов данных. Сформулирована суть несоответствия типов данных в разных ЯП. Семантика преобразования типов данных базируется на действующих подходах к их реализации в системах программирования с ЯП, влиянии архитектуры разных платформ и условий преодоления различий, как правило, путем создания промежуточной среды для поддержки связей разноязыковых компонентов и систем.

Источником формирования реализационной точки зрения в проблематике обеспечения взаимодействия разных программных объектов послужил анализ действующих системных сред (COM, CORBA, SUN, MS.NET, JAVA, Grid и др.) [26]. Общий подход к реализации названной проблемы — промежуточная среда (middleware), брокер запросов в распределенной среде, генерация необходимых посредников (stub, skeleton) в системе CORBA, DCOM, SUN и унификация выходного кода в системе MS.NET.

Сформулирована новая концепция, отличающаяся от существующих тем, что она основана на методе генерации общих типов данных GDT стандарта ISO/IEC 11404:2007 применительно к новым ЯП и гетерогенным средам. Рассмотрены особенности среды Grid и сформулированы пути подключения новых ЯП для этой среды. Создается библиотека функций преобразования GDT типов данных к соответствующим фундаментальным типам данных ЯП и библиотека функций для преодоления несоответствий их реализации в системах программирования и отличий платформ компьютеров по форматам данных и разрядности (32, 64) их памяти.

СПИСОК ЛИТЕРАТУРЫ

1. Пройдаков Е.М., Теплицький Л.А. Англо-український тлумачний словник з обчислювальної техніки, Інтернету і програмування. — К.: СофтПрес, 2006. — 824 с.
2. Чарнецки К., Айзенекер У. Порождающее программирование. Методы, инструменты, применение. — М.; СПб.; Харьков: Изд. дом «Питер», 2005. — 730 с.
3. Турский В. Методология программирования: Пер. с англ. — М.: Мир, 1981. — 265 с.
4. Агафонов В.Н. Типы и абстракция данных в языках программирования // Данные в языках программирования. — М.: Мир, 1982. — С. 267–327.
5. Замулин А.В. Типы данных в языках программирования и базах данных. — М.: Наука, 1987. — 152 с.

6. Лаврищева Е.М. Интерфейс в программировании // Проблеми програмування. — 2007. — № 2. — С. 126–139.
7. Лаврiщева К.М. Генерувальне програмування програмних систем і сімейств // Там же. — 2009. — № 1. — С. 3–16.
8. Грищенко В.Н. Метод объектно-компонентного проектирования программных систем // Там же. — 2007. — № 2. — С. 113–125.
9. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. Основы индустрии программных продуктов. — 2-изд., доп. и перераб. — Киев: Наук. думка, 2009. — 370 с.
10. Грищенко В.М. Теоретичні та прикладні аспекти компонентного програмування: Автореф. дис. ... д-ра фіз.-мат. наук. — Київ, 2007. — 35 с.
11. Лаврищева Е.М. Сборочное программирование. Теория и практика // Кибернетика и системный анализ. — 2009. — № 6. — С. 1–12.
12. Лаврищева Е.М. Методы программирования. Теория, инженерия, практика. — К.: Наук. думка, 2006. — 451 с.
13. Лаврищева Е.М. Становление и развитие модульно-компонентной инженерии программирования в Украине. — Киев, 2008. — 33 с. — (Препр. / НАНУ. Ин-т кибернетики им. В.М. Глушкова; № 1).
14. ETICS: the International software engineering service for the Grid / A. di Meglio, M.-E. Bégin, P. Couvares et al. // J. Physics: Conf. Series. — 2008. — **119** (http://dx.doi.org/10.1088/1742-6596/119/4/042010)**.**
15. Castelli D., Candela L., Pagano P., Simi M. DILIGENT: A digital library infrastructure for supporting joint research // Proc. of 2nd IEEE — CS Intern. Symp. on Global Data Interoperability — Challenges and Technologies, Sardinia, Italy, June, 2005 (http://ieeeplore.ieee.org/Xplore/).
16. ДСТУ ISO/IEC TR 10000-2:2004 Інформаційні технології. Основи та таксономія міжнародних стандартизованих профілів. Ч. 2. Принципи та таксономія OSI профілів (ISO/IEC TR 10000-2:1998, IDT).
17. Симкин С., Барлет Н., Лесли А. Программирование на Java. Путеводитель. — К.: DIASOFT, 1996. — 736 с.
18. Бей И. Взаимодействие разноязыковых программ. — М.; С.-Петербург; Киев: Изд. дом «Вильямс», 2005. — 868 с.
19. Шелдон Р., Мойе Д. MySQL: базовый курс = Beginning MySQL. — М.: Диалектика, 2007. — 880 с.
20. Хоар К. О структурной организации данных // Структурное программирование. — М.: Мир, 1975. — С. 92–197.
21. Электронная наука. Состояние проблемы. — Киев: ІПС НАН України, 2007. — 94 с.
22. Эммерих В. Конструирование распределенных объектов. Методы и средства программирования интероперабельных объектов в архитектурах OMG/CORBA, Microsoft/COM и Java/RMI. — М.: Мир, 2002. — 510 с.
23. C# 2008 и платформа .NET 3.5 для профессионалов: Пер. с англ. / К. Нейгел, Б. Ивьен, Д. Глинн и др. — М.: Изд. дом «Вильямс», 2009. — 1392 с.
24. Фомичев В.С., Пютчлер У. Промежуточные языки систем программирования // ЭВМ в проектировании и производстве. — 1987. — Вып. 3. — С. 251–270.
25. Лаврищева Е.М. Проблема интероперабельности разнородных объектов, компонентов и систем. Подходы к решению // Проблеми програмування. — 2010. — № 2–3. — С. 28–42.
26. Програмна інженерія: Підручник. — К.: Академперіодика, 2008. — 319 с.

Поступила 15.04.2010