УДК 681.03

ПРОБЛЕМА ИНТЕРОПЕРАБЕЛЬНОСТИ РАЗНОРОДНЫХ ОБЪЕКТОВ, КОМПОНЕНТОВ И СИСТЕМ. ПОДХОДЫ К ЕЕ РЕШЕНИЮ

Е.М. Лаврищева

Институт программных систем НАН Украины, тел. 526 3470

Введены понятия интероперабельности и ее базиса – фундаментальных типов данных. Определен основной объект рассмотрения – типы данных. Исследована проблема взаимодействия разнородных компонентов и систем – с теоретической, семантической и реализационной точек зрения. Представлена теория структурной организации типов данных, семантика преобразования нерелевантных типов данных и подходы к реализации типов данных языков программирования в современных средах (в том числе в распределенных средах типа Grid).

The concepts of Interoperability and Fundamental Data Types as its basis are introduced. An issue under consideration is defined to be the Data Types. The problem of heterogeneous components and systems interaction is investigated in theoretical, semantic and implementation perspectives. The theory of Data Types structuring, the semantics of non-relevant Data Types transformation and the approaches for programming languages' Data Types implementation in modern environments (including distributed ones like Grid) are given.

*«Я пытался представить, как можно подойти к этой проблеме с фундаментальной точки зрения. Я как будто нахожусь в центре сферы, а проблемы и возможно*сти окружают меня со всех сторон»

П. Хаггерти

Вступление

Интероперабельность – совместимость двух и более объектов. Данный термин имеет широкий спектр использования в социальной, деловой, производственной, учебной, компьютерной и программистской деятельности (например, взаимодействие: людей в обществе, бизнесе, работе, локальных, корпоративных и глобальных сетях и др.).

В информатике этот термин обозначает способность к взаимодействию двух и более компонентов или систем в целях обмена информацией и использования ее для организации вычислений [1, 2]. Взаимодействие имеет место: в языках программирования (ЯП) – аппарат подпрограмм и функций; в технических и программных средствах – интерфейс связи; в разноязыковых и разноплатформенных программах – модуль-посредник в языке MIL (Module Interconnection Language); в системах общего назначения (Sun IBM, Microsofts, CORBA, COM, JAVA и др.) – промежуточная среда брокера; а также в системах Linux и Windows Server и в средах (Microsoft.Net, IBM Web Spheke – средства Web Servises, единый выходной код и т. п. Оно предполагает совместимость программ и систем по структурам и типам данных, передаваемых через операторы вызова, запросов или протоколы. Нас более всего интересуют вопросы интероперабельности, связанные с программированием и использованием множества готовых программ (системных, прикладных, проблемных, научных и др.), с которыми специалисты различных предметных областей организуют взаимодействие при решении разного рода задач.

Готовые программы хранятся в многочисленных библиотеках (Matlab, IP, Demral и т.п.), в репозитариях e-science, в Интернете, в среде систем общего назначения, а также в фондах фирм и организаций разработчиков [3–7]. С общей точки зрения, такие программы – это готовые ресурсы (компоненты, сервисы, агенты и др.) для интеграции их в системы или семейства систем, а также для организации вычислений физических, математических, биологических и других задач в современных средах. При этом при организации вычислений с использованием современных компьютеров, мейнфреймов и кластеров возникают также проблемы несовместимости их платформ, используемых на них ЯП и готовых программных ресурсов, разработанных на основе фундаментальных структур и типов данных [8–13].

Особую актуальность приобретают еще и такие важные факторы.

1. Разнообразие принимаемых проектных решений при разработке больших программных систем (ПС) и без учета основных вопросов обеспечения совместимости их составных частей по ЯП и типам данных.

2. Сложность интеграции разнородных программ и компонентов повторного использования (КПИ) в рамках одной ПС обусловлена не только различиями аппаратно-программных платформ и выходного кода, произведенного соответствующими системами программирования в ЯП, а и принципами трансформации описаний предметных областей, конфигурирования компонентов ПС в семейства ПС, обеспечения их взаимодействия и жизнедеятельности в разных средах.

3. Распространение Интернет-технологий привело к созданию ряда новых языков описания данных и документов (XML, RDF, OWL и др.), которые также должны учитываться в новых разработках ПС при обеспечении задач передачи и обработки данных в новых сетевых средах, типа Grid [3, 14, 15].

Данный перечень не дает исчерпывающего представления обо всех проблемах программирования XXI века, которые связаны с обеспечением взаимодействия разноязыковых и разнородных программ в современных средах и сетях. Вместе с тем, продолжают появляться новые структуры данных (последовательности, графы, медийные объекты и др.) [6], подходы к генерации новых структур в выходной код для исполнения программ и унифицированные средства для формального описания общих типов данных в стандарте ISO/IEC 11404-2007 «General Data Types. Эти средства включают фундаментальные структуры и типы данных ЯП, а также механизмы генерации новых типов данных для обеспечения интероперабельности разнородных программ.

В данной работе рассматриваются следующие базовые основания интероперабельности [6, 24] и подходы к ее реализации.

1. Теория структурной организации типов данных ЯП как основа для взаимодействия программных структур.
2. Семантические аспекты интероперабельности разнородных программ.
3. Подходы к реализации типов данных на основе интерфейсов в современных средах.
4. Концепция генерации общих типов данных стандарта ISO/IEC 11404 и подходы к ее реализации в среде генерирующего программирования и в гетерогенных средах типа Grid [14, 15].

Далее излагается содержание каждого из приведенных разделов, объясняющих теоретическую, семантическую и прикладную сущность представления и реализации проблемы интероперабельности разноязыковых и разнородных компонентов и систем.

1. Теория структурной организации типов данных ЯП – основа интероперабельности в программировании

Теория структурной организации данных базируется на формализованном подходе к определению структур и типов данных с помощью аксиоматизации и правил выполнения операций над их значениями. Основу теории структурной организации данных [8–12] составляют типы, операции над ними и форма преставления на машине. Тип – это математическое понятие, обозначающее множество значений элементов. Базовый тип – это элементарный тип (например, целое, вещественное и др.), значение которого определяется аппаратурой, компиляторами программ с ЯП и др. Тип присваивается переменной, определяющей класс значений, каждое их которых принадлежит одному и только одному типу. Операции над значениями типа – это аксиомы. Каждое значение типа “строится” с помощью конечного числа операций (например, операция «+» имеет разную семантику для переменной, матриц и т.п.) и в памяти машины это значение – конечно.

Операции над типом – это операции преобразования для отображения значений одного типа в значения другого типа. Обратной для функции преобразования является селектор, т.е. функция, которая обеспечивает выбор компонентов из структурного или сложного типа.

Аксиоматическая 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 – это перечислимые типы данных, структурные – это массивы и записи, сложные типы – множества, списки, последовательности и др. [7–12].

Типы данных предназначены для описания функций и программ в ЯП. Они реализуются системами программирования на разных платформах компьютеров в выходном коде, который служит источником не только для выполнения программы на этом ЯП, но и для обеспечения интероперабельности (взаимодействия) в разнообразных, отличающихся между собой современных средах. Каждая реализованная программа отражает используемый тип данных конкретного ЯП, значение которого передается другой программе с помощью вызова (обращения) и используется при вычислении программы.

1.1. Аксиоматика простых типов данных. Простые типы – это перечислимые типы, а также “целый – integer (i)”, “вещественный – real (r)“, “булевский – boolean (b)“ и “символьный – character (c)“. Они имеют вид

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

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

Операции над перечисленными типами 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.

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

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

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

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

Операция ord придает каждому символу его порядковый номер, а char для этого номера – значение.

Аксиомы данных типов имеют такой вид:

$$X.\min \in X, \quad X.\max \in X,$$
$$(\forall x \in X) \& (x \neq X.\max) \Rightarrow \operatorname{succ}(x) \in X.$$
$$(\forall x \in X) \& (x \neq X.\max) \Rightarrow \operatorname{succ}(x) \neq X.\min,$$

где X.min и X.max – минимальный и максимальный элементы множества X.

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

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

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

Над переменными целого типа выполняются аналогичные операции, к ним добавляются операции целочисленной арифметики: унарный минус, +, -, ×, div и mod. Целый тип данных на заданном отрезке определяется следующими аксиомами:

$$X^i = \{X^i.\min, X^i.\max + 1, \ldots X^i.\max\},$$
$$\Omega^i = \{+, \times, \text{div}, -, \leq\},$$
$$\text{type } T^i = (X^i.\min, \ldots, X^i.\max).$$

Вещественный тип определяется с помощью операций сопоставления и обычных арифметических операций (унарный минус, +, –, ×, /). Аксиомы вещественного типа на отрезке таковы:

$$X^r = \{x \mid X^r.\min \leq x \leq X^r.\max\},$$
$$\Omega^r = \{+, \times, /, -, \leq\},$$
$$\text{type } T^r = (X^r.\min, \ldots, X^r.\max).$$

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 типу с помощью следующих аксиом:

$$(\forall x \in X) \Rightarrow T(T^0(x)) = x,$$
$$(\forall x_1 \in X) \& (\forall x_2 \in X) \Rightarrow (x_1 \leq x_2) \equiv (T^0(x_1) \leq T^0(x_2)).$$

В них 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 $\oplus$:

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

1.2. Аксиоматика структурных и сложных типов данных. Структурный тип данных – это массив и запись, строящиеся из базовых и неструктурных типов данных и содержащие набор упорядоченных элементов, каждый из которых обрабатывается как самостоятельный тип.

Массив конструируется из базовых перечислимых типов и представляется механизмом отображения множества индексов I массива на множество значений Y элементов массива [8]:

$$M: I \rightarrow Y.$$

Над данным типом – массив – 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

$$X^\alpha = \{x \mid (\forall x_1 \in X^\alpha) \& (\forall x_2 \in X^\alpha) \Rightarrow I(x_1) =$$
$$I(x_2)) \& (Y(x_1) \cup Y(x_2) \subset \overline{Y}(X^\alpha))\}, \ \Omega^\alpha = \{\leq\},$$
$$\text{type } T^a = \text{array } T(I) \text{ of } T(\overline{Y}),$$

где $I(x)$ – множество индексов x для массива, $Y(x)$ – множество значений элементов массива, $\overline{Y}(X^\alpha)$ – множество значений элементов для любых типов массива, T^a – тип данных *массив;* $T(I)$ – тип данных индексов массива; $T(\overline{Y})$ – тип данных для множества значений элементов массивов типа T^a.

Исходя из второго выражения приведенного определения к данному типу принадлежат 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, соответствующим типу данных массива. Данные типа $T(Y)$ в описании массива T^a в свою очередь могут определяться рекурсивно, т.е. через массив по следующей схеме: type T^a = array $T(I^1)$ of $T(Y^1)$,

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

что эквивалентно следующей записи: type T^a = array $T(I^1 \times I^2)$ *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, как и массив, представляет собой конкатенацию отдельных компонентов, которые могут иметь разные типы. Множество значений типа *запись* – прямое произведение множества значений ее компонентов. К множеству операций, выполняемых над записями, относятся только операции сопоставления, используемые для сравнения однотипных структур (типы компонентов сравниваемых записей и их порядок следования одинаковы):

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

Общая форма представления типа для записи имеет вид

$$\text{type } T^z = (S_{v1}{:}T^{v1}; \ldots S_{vn} : T^{vn}).$$

где S_{v_1}, ..., S_{v_n} – селекторы, а T^{v_1}, ..., T^{v_n} – типы данных компонентов записи.

В ЯП запись $(S_{v_1}, \ldots, S_{v_n})$ описывается следующим образом:

$$\text{type } T^z = \text{record}$$
$$S_{v_1} : T^{v_1}, : S_{v_n} : T^{v_n} \text{ end.}$$

Операции выполняются над записью как над единым структурным значением. Обработка отдельных компонентов записи выполняется с помощью операций селектора, аналогично соответствующей операции для обработки массива.

Сложные типы данных. К ним относятся: множества, объединения, списки, последовательности, деревья и др. Некоторые из этих типов – стандартные в конкретных ЯП, другие реализуются путем моделирования через соответствующие структуры и операции над ними. При этом представление данных операций в ЯП менее формализовано.

Множества. Общая форма представления этого типа данных имеет вид:

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

где T – тип множества; T^0 – базовый тип для его элементов.

Тип T включает операции над множествами как над математическими типами – объединение, пересечение, разность, включение, тождественность и др. С помощью операций селектора производится 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.

Объединения. Общая форма для типа – объединение имеет вид:

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

где T – тип объединения; T^{v_1}, ..., T^{v_n} – базовые типы.

Любой объект типа T имеет значение и признак, по которому определяется один из типов T^{v_1}, ..., T^{v_n} для данного значения. Механизм реализации объединения подобен механизму реализации вариантных записей. Отличие состоит в том, что сам признак скрыт в отличие от признака вариантной записи, в которую он входит в качестве отдельного компонента.

Списки. Это конструктивный элемент ЯП Лисп. Элемент списка описывается как запись с одной или несколькими компонентами ссылочного типа, над которыми выполняются операции, аналогичные операциям над фиксированными записями. Существуют также операции, применяемые к целому списку: выбор начального элемента, получение остатка списка, соединение списков, их сравнение, инвертирование, поиск элементов в списке и др. Эти операции принадлежат к средствам языка Лисп.

Последовательность. Общая форма имеет вид: type T = sequence T^0, где T – тип последовательности, а 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. Одна из разновидностей последовательности – строка. Для строки каждый элемент, кроме ссылочной переменной, содержит элемент символьного типа.

Деревья. Это списковые структуры, которые используются для представления графов или других аналогичных объектов. Множество операций над деревьями аналогично множеству операций над списками. Реализация этих операций зависит от конкретных приложений.

Кроме рассмотренных типов, в программировании используются таблицы, файлы и всевозможные комбинации вышеперечисленных типов.

2. Семантические аспекты интероперабельности разнородных программ

Проблема интероперабельности с семантической точки зрения – это обеспечение совместимости типов данных с помощью операций преобразования форм представления типов данных в программах ЯП к релевантному или единому программному коду среды в случае некоторых отличий в описании типа переменной в операторах вызова программ друг другом. Кроме того, базовый тип данных в разных ЯП может быть неадекватно реализован соответствующими системами программирования и связанными с ними платформами. На формальном уровне речь идет об изоморфном преобразовании типов данных к виду, подходящему для вызываемого компонента и среды выполнения.

Случаи изоморфного преобразования типов данных.

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. Изоморфное отображение не существует между множествами значений типов данных разных ЯП.

Изоморфизм отображений множества значений t типов данных T с помощью операций Ω_a^t определяется мощностью системы преобразования, под которой понимается мощность множества X_a^t [6]. Данный тривиальный результат может служить сильным критерием оценки правильности взаимодействия программ для случая, если в разных ЯП одинаковые типы данных имеют различные множества значений. Другим применением этого критерия может служить перенос программных систем из разноязыковых компонентов с компьютера одного типа на компьютер другого типа, причем эти типы имеют различные архитектуры и диапазоны целых и вещественных чисел.

Анализируя множества операций Ω_a^t, можно отметить, что они все содержат операции сопоставления. Поэтому на самих множествах X_a^t задано отношение порядка. Учитывая, что любые два элемента из X_a^t сравнимы, это отношение определяет линейный порядок. В этом случае изоморфное отображение должно сохранять отношение линейного порядка.

Пусть $\sum$ – множество алгебраических систем, построенных выше. Тогда имеет место следующая лемма.

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

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

Доказательство данной леммы простое. Для всех построенных систем преобразования, описывающих простые типы данных, множества значений ограничены. Согласно аксиоме, минимальные и максимальные элементы этих множеств принадлежат им. Структурные типы этих данных строятся из простых с помощью конечного числа операций, а их множества значений также конечны. Поэтому $X_{a.\min}^t$, $X_{a.\max}^t$, $X_{\beta.\min}^q$, $X_{\beta.\max}^q$ существуют и принадлежат множествам X_a^t и X_β^t соответственно. Учитывая линейную упорядоченность этих множеств, необходимо, чтобы выполнялось условие леммы. Если бы оно не выполнялось, то изоморфное отображение не сохраняло бы линейный порядок, что противоречит сделанному ранее выводу. Лемма доказана.

Системы изоморфного отображения между основными типами данных (перечислимый тип – в перечислимый; булевский – в булевский; целый – в целый; вещественный – в вещественный) рассматривать не будем в силу их простоты. Однако возможны более сложные преобразования между следующими типами данных: символьный – в целый; символьный – в булевский; целый – в символьный; целый – в булевский; булевский – в целый; булевский – в символьный. Эти виды семантического преобразования представлены и им дано доказательство в [6].

При выполнении операций преобразования массивов и записей должен сохраняться только линейный порядок. Возможны следующие виды преобразований: массив – в массив, запись – в запись, массив – в запись, запись – в массив.

Рассмотрим, например, первое из них. Пусть φ – их изоморфизм, сохраняющий линейный порядок. Это означает, что

$$(\forall\ x_{\alpha_1}^a \in X_\alpha^a)\ \&\ (\forall\ x_{\alpha_2}^a \in X_\alpha^a)\ \&\ (x_{\alpha_1}^a \le x_{\alpha_2}^a) \Rightarrow \varphi(x_{\alpha_1}^a) \le \varphi(x_{\alpha_2}^a).$$

Здесь $x^a_{\alpha_1}$ и $x^a_{\alpha_2}$ – функции отображения. Выполнение отношения ≤ для функций означает выполнение этого отношения для всех элементов области определения данных функций. Из выражения для функций φ($x^a_{\alpha_1}$) и φ($x^a_{\alpha_2}$) следует:

$$x^a_{\alpha_1} : I^a_\alpha \to Y(x^a_{\alpha_1}),\ Y(x^a_{\alpha_1}) \subset \overline{Y}(X^a_\alpha),$$
$$x^a_{\alpha_2} : I^a_\alpha \to Y(x^a_{\alpha_2}),\ Y(x^a_{\alpha_2}) \subset \overline{Y}(X^a_\alpha).$$

Пусть выполняется отображение:

$$\varphi_i : I^a_\alpha \to I^a_\beta, \qquad \varphi_y : \overline{Y}(X^a_\alpha) \to \overline{Y}(X^a_\beta),$$

где φ_i и φ_v – изоморфные отображения, соответствующие множествам индексов i и значениям y для элементов массива. Обозначим E_1 вложение $Y(x^a_{\alpha_1}) \to \overline{Y}(X^a_\alpha)$, а E_2 – вложение $Y(x^a_{\alpha_2}) \to \overline{Y}(X^a_\alpha)$. Тогда φ($x^a_{\alpha_1}$) и φ($x^a_{\alpha_2}$) будут определяться как ограничения отображения φ_y на соответствующие подмножества и обозначаться $\varphi_y|Y(x^a_{\alpha_1})$ и $\varphi_y|Y(x^a_{\alpha_2})$, соответственно. Последнее неравенство будет эквивалентно $\varphi_y \mid Y(x^a_{\alpha_1}) \le \varphi_y \mid Y(x^a_{\alpha_2})$.

Рассмотрим далее преобразование типа запись – в другую запись. В теории структурной организации данных множество значений записей определяется как прямое произведение множеств значений их отдельных элементов. Данный подход к описанию записей представлен на примере множества значений для трех описаний элементов записи а), б), в):

а) type $T^{z_1} = (S_{v_1} : T^{v_1};\ S_{v_2} : T^{v_2};\ S_{v_3} : T^{v_3})$,

б) type $T^{z_2} = (S_{a_1} :\ T^{v_1};\ S' : T')$, type $T' = (S_{v_2} : T^{v_2};\ S_{v_3} : T^{v_3})$,

в) type $T^{z_3} = (S' : T';\ S_{v_3} : T^{v_3})$, type $T' = (T_{v_1} : T^{v_1};\ S_{v_2} : T^{v_2})$.

Описания трех типов записей T^{z_1}, T^{z_2}, T^{v_3} отличаются, их множества значений совпадают в плане задачи преобразования их типов данных. Между множествами типов данных записей можно установить взаимно однозначное соответствие при двух условиях:

– количество компонентов в записях одинаково;

– для каждого из элементов первой записи существует только один соответствующий элемент второй записи.

Операции сопоставления над записями выполняются с учетом выбранного соответствия, и преобразование записи сводится к отображению типов отдельных элементов. Если изоморфные отображения между отдельными элементами сохраняют линейный порядок, то преобразование изоморфно.

Впрочем, преобразование «запись – в массив» требует, чтобы типы всех элементов записи были одинакового типа, а сама запись была представлена в виде массива. Множество значений элементов записи образует множество значений элементов массива *Y*. Множество индексов переупорядочивает компоненты записи: *i*-й компонент записи ставится в соответствие φ(*i*)-у элементу во множестве *J* (φ – взаимнооднозначное отображение). Само множество *I* – множество значений перечислимого типа.

3. Подходы к отображению типов данных в интерфейсах взаимодействия разнородных объектов в современных ЯП и средах

К настоящему времени создан ряд операционных сред общего назначения (ONC SUN, OSF DCE, COM, SOM, CORBA, JAVA, MS.Net и др.) [3, 6,17, 18 и др.] с учетом стандарта открытых систем OSI (Open Systems Interconnection) [19]. Каждая среда обеспечивает обработку (связь, взаимодействие, выполнение и др.) разнородных программных объектов со своими специфическими особенностями.

Модель OSI задает стандартный механизм преобразования форматов данных, передаваемых по сети с одной платформы на другую (например, SUN, VAX, IBM, INTEL) путем кодирования (code) или декодирования (decode) на основе стандартных соглашений для представления символов (ASCII, EBCDIC и др.), форматов целых чисел и чисел с плавающей точкой (IEEE, VA, IBM и др.), формата символьного типа ISO Latin/1 и др. Каждой платформе соответствует своя методика выравнивания данных по длине для размещения значений базовых типов и сложных структур данных. Каждая среда имеет дополнительные инструменты для преобразования данных, специфицированных в ЯП и IDL [3, 6, 12].

Для устранения неоднородности переданные данные кодируются, т.е. преобразуются к общему виду CTS (common transfer scheme) промежуточной среды с учетом представления, прикладного и платформенного уровня модели OSI. Приведенные среды общего назначения организованы таким образом, чтобы программы на разных ЯП могли в них выполняться и обмениваться данными через :

– операторы обращения к локальным подпрограммам в ЯП;

– операторы удаленного вызова (RPC, RMI) или запроса в языке IDL для систем, распределенных по разным компьютерам в сетевой среде;

– протоколы (IIOP, GIOP, TCP/IP и др.) передачи данных от клиента к серверу;

– промежуточные среды (языковые, семантические, системные и др.);

– стандартные языки описания данных и документов в сетевой среде (XML, RDF, OWL и др.).

Фундаментальные типы данных (ФТД), которые используются при описании программ разными ЯП, преобразуются к формату данных в среде выполняющей платформы для получения результата, а затем “обратно” – к исходному формату.

Далее рассмотрены специфические особенности обеспечения взаимодействия разнородных программ в названных средах.

3.1. Взаимодействия компонентов в системах ONC SUN, OSF DSE, DCOM и CORBA. Взаимодействие компонентов в этих системах основано на удаленном, т.е. в посреднике stub, операторы которого обеспечивают передачу данных по сети, вызове RPC интерфейса. Преобразования данных связаны с их форматом, некоторыми различиями в архитектуре машин и в кодах различных компиляторов, которые выполняют отображение базовых и нерелевантных типов данных с устранением неадекватности элементов выходного кода программ в ЯП и преобразованием их в промежуточный код распределенной среды [4].

Объектная модель DCOM устанавливает связь распределенных объектов в системе CORBA с помощью брокера ORB через запросы и описания посредников: stub – для клиента и stub/skeleton – для сервера. Объекты описываются в современных объектно-ориентированных ЯП. Формализмы обеспечения взаимодействия объектов и разноязыковых компонентов в системе CORBA таковы:

– механизмы передачи запросов удаленным объектам через stub и skeleton;

– обмен данными через сеть и их преобразование согласно форматному представлению платформ компьютеров среды;

– процедуры преобразования типов данных для каждой пары ЯП (С↔Смолток, Смолток ↔ Ада, Ада ↔ Кобол, Кобол ↔ Java, Кобол ↔ Ада и др.).

В системе DCOM реализованы следующие формализмы:

– механизмы передачи данных (типа RPC-вызов) и описание интерфейса;

– сетевой обмен данными;

– обмен данными и преобразование нерелевантных типов данных путем кодирования и декодирования данных, передаваемых с разных архитектур компьютеров. Процедуры преобразования данных для сред ONC, DCE и CORBA реализованы на языке C++. Интерфейс описывается в языке IDL.

Типы данных ЯП разделены на две группы: базовые и сложные. Базовые типы – это фундаментальные типы, а к сложным относятся interface, struct, union, sequence и array. Тип struct аналогичен описанию в языке C++; sequence и array содержат элементы одинакового типа переменной и фиксированной длины соответственно; тип union семантически соответствует типу union в языке C++. Спецификация параметров компонентов для распределенной среды CORBA такова: IN – входные, OUT – выходные, INOUT – результат.

В среде CORBA преобразование сложных типов данных осуществляется с помощью функций отображения типов, описанных спецификациями IDL. Трансформация типа struct включает последовательное преобразование всех полей, в порядке, указанном в спецификации IDL. Эту функцию компилирует IDL, порождая файлы отображения в соответствующие конструкции языка C++ и процедуры обращения к сервису брокера ORB. Функции преобразования базовых типов обеспечивают выравнивание данных, а также отображение сложных типов, имеющих вложенную структуру. Тип данных array преобразуется процедурами с использованием простых типов данных.

Определение типа с переменной длиной – рекурсивное, оно влечет за собой изменение «fixed или variable» для составных типов. Для клиента и сервера, использующих параметры OUT и RESULT выполняется перепись типов в тексте программы. Для сложных типов данных введен специальный тип – указатель (T_var), через который передаются все параметры для объекта сервера. Многие вопросы, связанные с типом string, решаются непосредственно пользователем.

Таким образом, подводя итоги анализа интероперабельности, отметим один общий подход для связи разноязыковых программ, основанный на генерации необходимых для них промежуточных компонентов (stub, skeleton) с помощью брокера объектных запросов в распределенной среде CORBA, DCOM и SUN.

3.2. Взаимодействие компонентов в среде Java. В системе JAVA реализован новый подход к обеспечению взаимодействия разных типов компонентов, основанный на механизме удаленного метода RMI, который дополняет язык JAVA стандартной моделью EJB (Enterprise Java Beans) компании SUN. К ней подключены классы языка JAVA, определения их атрибутов, параметров среды и свойств группирования компонентов в прикладную программу для выполнения на виртуальной машине JVM. Механизм развертывания JAVA-компонентов типа beans на сервере реализован для программ в исходном языке в среде модели EJB [17].

Для реализации и повторного использования компонентов типа beans в системе разработаны следующие шаблоны в языке Java for Forte:

– Beans для создания нового компонента и формирования каркаса компонента с простыми свойствами и возможностью автоматического изменения этих свойств;

– BeanInfo для интеграции beans компонентов и обеспечения взаимодействия;

– Customizer для создания панели, на которой размещаются элементы, которые со временем можно использовать для управления конфигурацией beans компонентов;

– Property Editor для создания класса, который используется во время проектирования и редактирования свойств beans компонентов.

Система обеспечивает связь языков Java и C++, основаную на преобразовании Java классов, библиотеке классов C++ и языке описания интерфейса Java Native Interface. Основа взаимодействия компонентов в этих языках – аппарат установления соответствия класса интерфейса в C++ с Java-классом интерфейса. Транслятор C++ продуцирует Java-прокси в C++ класс, результат которого – код Java–C++ является интероперабельным.

Реализация системы конвертирования релевантных типов данных между разноязыковыми компонентами (Java↔C++, C++↔Java и др.) основывается на дополнительных описаниях компонентов, соответствующих доработках компиляторах с учетом этих ЯП и особенностей платформ расположения компонентов.

3.3. Обеспечение взаимодействия компонентов на платформе MS.NET. Платформа MS.NET предоставляет интегрированную среду для поддержки *многоязыковой разработки* ПС в разных ЯП (C#, C++, VBasic, JAVA, Pascal, Perl, PHP, Prolog, Ruby, Smalltalk и др). Она поддерживает такие возможности [6]:

– взаимодействие с существующим кодом компонента;

– межязыковое взаимодействие с наследованием, обработкой исключительных ситуаций и отладкой;

– использование одного и того же набора встроенных типов данных CTS (Common Type System) для всех ЯП;

– библиотеку базовых классов, объектную модель для всех ЯП платформы .NET.

Для программ на ЯП создается двоичный файл, который является платформо-независимым «промежуточным языком» MSIL (Microsoft Intermediate Language) и обеспечивает процесс взаимодействия выходных с любых ЯП. Набор MSIL ассемблируется и может выполняться на разных архитектурах и платформах данной среды. Файлы с расширениями exe или dll – это программный код на языке CIL с дополнительными служебными метаданными о типе сборке, версии, ссылках на внешние компоненты и т. п. Перед своим выполнением такие файлы проходят определенную настройку для обеспечения работы в условиях конкретной платформы с помощью JIT-компиляторов (Just-In-Time compilers) среды CLR. CIL-код используется для перевода на промежуточный язык и машинный (native) платформы выполнения. Компилятор программ в ЯП создает файл на языке CIL, который ассемблируется (assembly) в сборный и переносной (Portable Executable) код.

Основной базовый компонент среды .NET – (Common Language Runtime), предназначенный для выявления и загрузки типов .NET, исходя из стандартной системы типов CTS, которая вместе с языком CLS (Common Language Specification) предоставляет средства описания всех типов данных для обеспечения взаимодействия с другими типами, которые будут представлены в форме метаданных .NET. К списку типов данных языка спецификации CLS отнесены стандартные типы данных, а также: классы, структуры, интерфейсы членов типов, перечислимые типы, делегаты и т. п. Значительное количество классов библиотеки FCL можно классифицировать в *пространстве имен* (Namespace). Например, в пространстве в Windows.Forms можно задавать формы окон, на которых размещаются элементы управления. Типы CTS могут использоваться после инициализации (с учетом метода вызова, операций get и set и т.д.), преобразовываться и объединяться в пространства имен System.

Универсальная система типов в Microsoft .NET представлена в виде иерархии. В ней выделены две группы типов: *типы-значений (*value type) *и типы-ссылок (*reference type), которые для всех ЯП задаются пользователем в программе на ЯП. В системе обеспечивает механизм отображения типов CTS в типы конкретных ЯП и наоборот.

Типы-значения – это статические типы, память для которых выделяется в стеке и освобождается после завершения работы программы. Эти типы встроены в CTS, их значения могут занимать память от 8 до 128 байтов, не принимают участия в наследовании и копируются при присвоении им значений.

Типы-ссылки или ссылочные типы используют указатели на объекты, которые они типизируют, а также механизмы централизованного хранения и освобождения памяти. Объекты этого типа – динамические, память под них выделяется из «кучи» и освобождается после уничтожения, «уборки мусора».

Ссылочные типы включают: объектные типы (object type); интерфейсные типы (interface type); типы-указатели (pointer type).

Эти типы выступают в роли спецификации формальных параметров, а интерфейсный тип обеспечивает связь разноязыковых компонентов.

3.4. Практический подход в обеспечению взаимодействия разноязыковых компонентов И. Бея. На множестве современных ЯП (C/C++, Visual C++, Visual Basic, Matlab, Smalltalk, Lava, LabView, Perl) были проверены и сформулированы практические способы обеспечения взаимодействия пар программ в этих ЯП в работе И. Бея [20].

Основным подходом к решению проблемы взаимодействия программ – технология доработки до каждой пары взаимодействующих программ дополнительных программных фрагментов, в которых по операторам вызова программируются конкретные операции преобразования типов передаваемых данных для передающей и принимающей платформы, что аналогично методу генерации модулей-посредников в систем АПРОП [6]. То есть здесь тоже главное связывающее звено при связи разноязыковых программ – оператор вызова, реализация которого выполняется разработчиком за терминалом с использованием всех современных визуальных способов конструирования программных посредников. Однако такой подход оправдан при статическом исполнении работ, как механизм проверки, но для динамики без присутствии разработчика требуется автоматизированная генерация разных аспектов взаимодействия. Далее коротко приведены специфические особенности обеспечения взаимодействия пар разноязыковых программ в разных средах.

Visual Basic и ЯП (C, C++, API Viewer и Window) связываются с помощью оператора обращения, параметрами которого могут быть текстовые строки, значения, массивы и другие типы данных. При передачи значений типов данных для их совместимости используются функции Windows API, API, DLL и

дополнительные операции их преобразования в случае некоторого несоответствия данных. Примером служит реализация Интернет-приложения с использованием HTML-страниц в Basic Visual и их размещение в Веб-браузере. *Matlab* – инструментарий для решения задач линейной и нелинейной алгебры, операций над матрицами и обеспечивает математические вычисления с помощью Matlab Compiler, Matlab C++, Matlab Libriary, Matlab Graphic Library. Применение этого инструмента основано на MatlabCompiler, который по запросу преобразует программы из формата Matlab (т. е. М-файлы или М-функции) в формат языка C++. Сформированный файл вызывается из программы в C++ и данные преобразуются к виду архитектуры принимающего и отправляющего компьютера.

Smalltalk обеспечивают создание приложений из программ в этом языке и в языках C, C++ и Matlab в среде VisualWorks. Создается модель приложения с методами реализации объектов и сообщениями для передачи значений внешним объектам. Модель использует из внешнего интерфейса функции DLL, а также функциями из библиотеки C++ для обеспечения совместимости типов передаваемых между разными программами данных.

LabView – инструмент для автоматизации производственных процессов, сборки данных, проведения измерений и управления разработкой комплекса программ реального времени в ЯП C, C++, Visual C++, Basic., которые взаимодействуют с драйверами аппаратуры, а также обеспечивают тестирование и выполнение программ с пульта. Система взаимодействует с ANS C, Visual Basic, Visual C++ Lab Windows/CV. Роль функций связи – измерение реальной аппаратуры типа термометры, переключатели и т.п.

Java обеспечивает взаимодействия со всеми ЯП – C, C++, Visual C++ и Matlab. Схема взаимодействия языков апробирована в среде виртуальной машины JAVA, реализация которой основана на использовании интерфейсных функций C, C++ и платформоориентированной библиотеки C++ при преобразовании передаваемых типов данных между программами на этих языках.

Язык Perl – сценарный. Он предоставляет средства описания сценариев взаимодействия с Интернетом, управление выполнением задач и созданием CGI-сценариев для сервера в системе Unix. Имеет интерфейс с языками C, C++, Visual Basic и Java. Интерпретатор языка Perl реализован на языке C и каждый интерфейс с другим ЯП рассматривается как расширение процедур динамической библиотеки C++. Оператор вызова программ в C или C++ обеспечивает его преобразование в специальный код, который размещается в библиотеке интерпретатора Perl, который может быть включен в Win32.

Таким образом, рассмотренные примеры взаимодействия практически проверены и ими можно пользоваться как готовыми шаблонами с принципами взаимодействия разноязыковых компонентов, настраиваемых для конкретной среды.

4. Концепция генерации общих типов данных стандарта ISO/IEC 11404 для будущих гетерогенных сред

4.1. Характеристика общих типов данных стандарта. Для обеспечения взаимодействия между разнородными программными объектами, реализованными в ЯП для современных сред и будущих сред, разработан стандарт ISO/IEC 11404–2007, который предоставляет формальный математический аппарат спецификации типов данных в общем виде, который независим от синтаксиса описания фундаментальных типов данных в ЯП. Данный 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 содержит формальные механизмы задания параметров интерфейса в языках (IDL, API, RPC) при вызове или обращении к процедурам, системам, компонентам повторного использования (КПИ), 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 описание, спецификатор типа данных, его значение в пространстве значений и операции над типами данных. Общий тип данных включает такие понятия:

- концептуальное или абстрактное понятие, которое его характеризует по значению и свойствам;
- структурное понятие, которое характеризует этот тип данных как концептуальный, стандартный интерфейс к сервису;
- реализационное понятие, которое определяется правилами представления типа данных в заданной среде.

К абстрактным нотациям относятся примитивные и не примитивные типы данных, базирующиеся на понятии структурной организации [9–12], которые используются в описании фундаментальных типов данных ЯП и интерфейсов программ. Структурные нотации используются для спецификации сложных типов данных и словарей реализации типов данных, отличающиеся от концептуальных понятий способом 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. В виду разнообразия типов данных и их нотаций, системы должны: обеспечивать описание и преобразование внутренних типов данных к типам данных стандарта или ЯП, т.е. они должны основываться на:

– описании типов данных, механизмов их генерации, а также нотации и значений из пространства значений, оговоренных стандартом;

– операциях над общими типами данных, характеристических операциях для организации работы с типами данных стандарта;

– перечне типов данных стандарта, для которых обеспечиваются стандартные внутренние и внешние преобразования.

Данный стандарт будет поддерживаться другими стандартами, которые формализуют средства преобразования типов данных в стандарте языка к промежуточным (параметризованным) типам данных, в том числе и к ЯП. Для этого в стандарте имеются механизмы перехода от параметризованного типа к общему. При этом параметрические значения определяются пользователями или разработчиком стандарта. Разработчик члена семейства СПС определяет параметризированные параметры для их применения и преобразования. Например, в стандарте языка определяется тип данных integer, а соответствующий процессор реализует некоторый диапазон этого целого – 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.

4.2. О принципах взаимодействия систем семейства в среде генерирующего программирования и новый подход к их реализации. В последние годы быстро развивается генерирующее программирование (Generative programming). Оно объединило в себе все современные парадигмы программирования (ООП, КП, сервисное, аспектное, композиционное, функциональное, ментальное и др.) и его называют мульти-парадигменным программированием [3, 13]. Ко времени его появления расширился набор и повысилась сложность новых объектов программирования – прикладные системы (applications), домены (domains), семейства систем (families systems) и их совокупности, которые определяются разными средствами этих парадигм. В данном программировании естественно главный стержень совместного использования продуктов разных парадигм – генерация выходного кода с обеспечением взаимодействия членов семейства в процессе их выполнении в некоторой операционной среде.

Каждая предметная область (ПрО) семейства имеет свою терминологию, понятия, известные специалистам, которые с ней будут работать, а также особенности, характеристики и принципы обмена данными при взаимодействии готовых программ и систем. Главный целевой продукт в генерирующем программировании (ГП) – семейство систем для заданной ПрО или домена. В нем каждый член может создаваться из готовых КПИ или его генерировать с помощью общей генерирующей модели GDM (Generative Domain Model), в которой специфицированы структуры, характеристики, свойства понятий и типов данных ПрО.

Ключевые понятия парадигмы ГП – *пространство проблемы, пространство решений* и *генерирующая модель* GDM, которая отображает пространство проблемы в пространство решений (рис. 1).

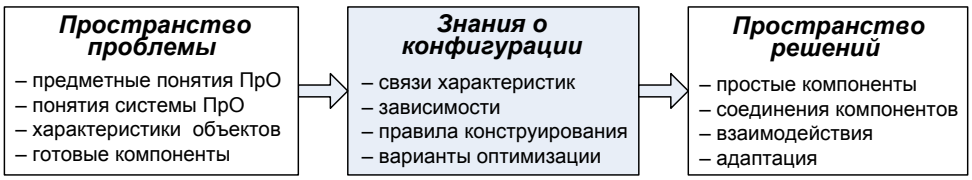

Рис. 1. Структура модели GDM

Пространство проблемы – это абстрактные понятия ПрО, характеристики (проектные решения, правила взаимодействия членов семейства и т. п.), а также подобранные готовые компоненты и КПИ для некоторых функций системы. Абстракции этого пространства входят в спецификацию компонентов в ЯП и трансформируются в объекты пространства решений. Они могут быть уточнены или изменены в зависимости от языка описания домена. Для представления модели ПрО разработан абстрактный механизм описания специфики домена – DSL (Domain Specific Language). Преобразование пространства проблемы в пространство решений осуществляется конструктивными правилами связи абстракций и понятий обеих пространств (через ссылки, запросы и др.) для их представления в конфигурационной базе, используемой и при решении задач ПрО.

Пространство решений – это множество программных абстракций (компоненты, каркасы, шаблоны, связи, аспекты защиты или взаимодействия и др.) и готовых к выполнению компоненты, которые могут быть простой или сложной природы в зависимости от смысла задач и функций домена. Связь между этими пространствами осуществляет модель GDM, описание которой ПрО в языке DSL трансформируется в конкретный ЯП или несколько ЯП. Специфические особенности ПрО в базе конфигурации используются при конфигурировании компонентов в систему.

Генерация СПС реализуется в интегрированной среде разработки (IDE – Integrated Development environment) с использованием двух подходов (рис. 2). *Конфигурационный подход* обеспечивает интеграцию из готовых разнородных компонентов повторного использования (КПИ) одиночных программ (Application Engineering), построение членов семейства или самого СПС из готовых приложений, КПИ и систем. Разработку членов СПС из КПИ, которые хранятся в специальном хранилище – репозитарии интегрированной среды, можно назвать *конвейерной*. Каждый компонент репозитария должен специфицироваться таким образом, чтобы было известно все о параметрах связи с другими КПИ, в том числе о передаваемых данных, их типах и структурах, отображаемых в форматы среды *взаимодействия* разных компонентов.

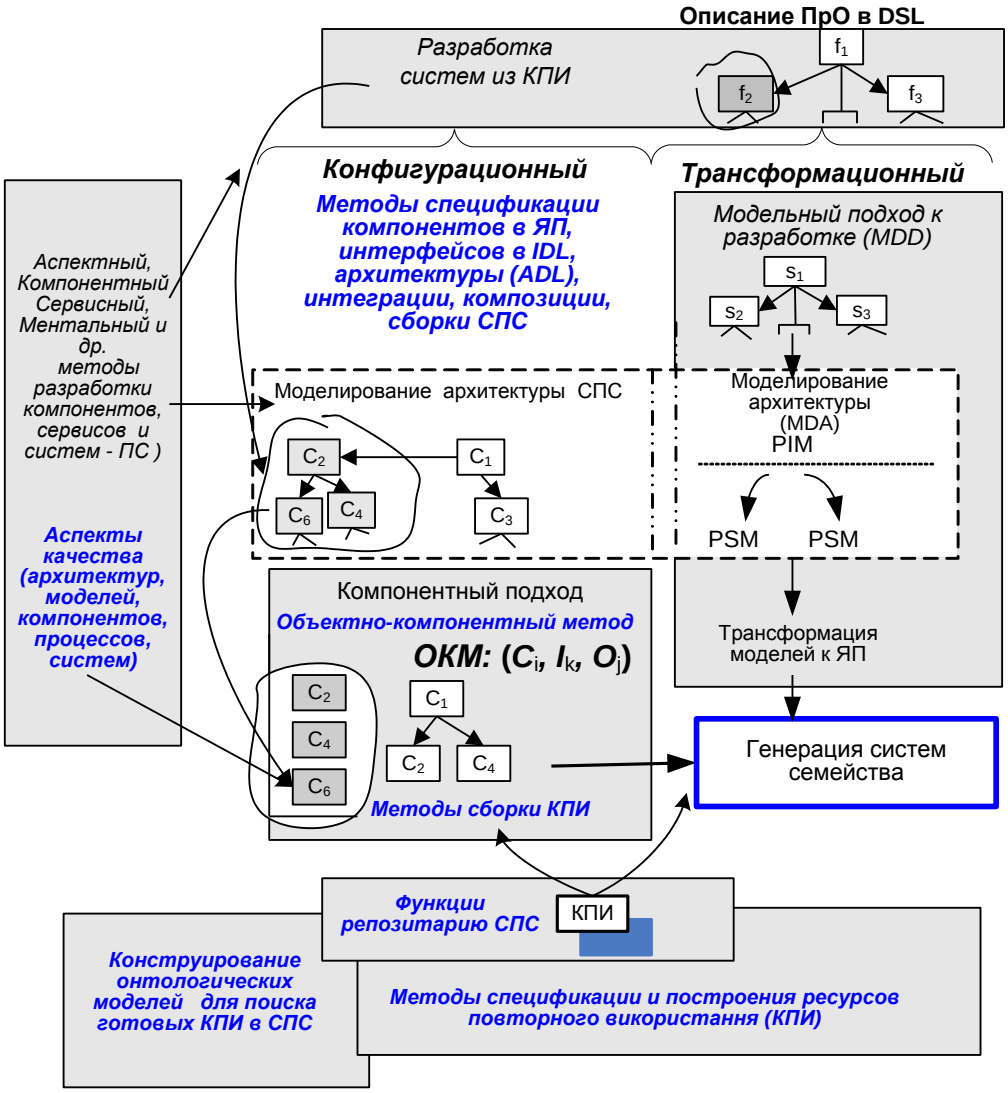

Рис. 2. Два подхода к созданию СПС

Трансформационный подход предполагает последовательную разработку членов семейства, начиная с требований, формального описания модели GDM, процессов отображения ее объектов к некоторому промежуточному состоянию артефактов, пополняя ими пространство решений. Отображение модели GDM определяется двухуровневой архитектурной моделью реализации ПС, которая может быть платформо-независимой – модель PIM (Platform Independent Model) и платформо-зависимой – модель PSM (Platform Specific Models). Архитектуры MDA (Model Driven Architecture) моделируется путем отображения PIM в PSM, согласования методов генерации моделей прикладных систем (Application model) с описанием их DSL-языков или ЯП.

В ГП отсутствует целевая промежуточная среда для поддержки разных объектов мультипарадигм. Имеются отдельные инструменты, например, для поддержки аспектного программирования (AspectJ), ведения готовых компонентов в библиотеках IP, GMCL и Demral и др.

В фундаментальном проекте ИПС НАН Украины “Разработка теоретического аппарата ГП и интегрированной среды ее поддержки” (2007–2011 гг.) ставится цель – разработать такую промежуточную инструментальную среду, которая обеспечивала бы реализацию базовых концепций ГП по созданию СПС, включая новые концепции формального описания СПС, тестирования членов семейств, экспертиз процессов, оценивания качества КПИ и обеспечения жизнедеятельности как отдельных членов СПС, так и СПС.

Инструментальная среда ГП должна обеспечить взаимосвязь языковых, технологических и прикладных инструментов средств поддержки ГП, реализацию конфигурационного подхода к созданию СПС из готовых разнотипных и разноязычных КПИ мультипарадигмы ГП. Главная особенность данного научного проекта – генерация разнородных КПИ, с обеспечением интероперабельности типов данных систем и их переносности в другие гетерогенные среды, типа Grid [14, 15].

Построенная нами теория компонентного программирования [7, 21] образует базис теории ГП. Внешняя и внутренняя алгебра этой теории будет расширена новыми операциями, обеспечивающими преобразование GDT типов данных разноязычных компонентов к фундаментальным типам структур и данных ЯП (и обратно), как базис интероперабельности компонентов и систем. Проводиться работа по созданию инструментария преобразования GDT к ФТД ЯП (на примере трех базовых ЯП – JAVA, C++, Pascal) с участием студентов 4–5 курса факультета кибернетики КНУ им. Тараса Шевченко, которые прослушали курсы лекций по технологии программирования и программной инженерии [6, 13–16, 22–25], включая анализ проблемы взаимодействия разноязыковых компонентов в раннее упомянутых (п. 3) современных средах, и выполняли дипломные работы, связанные с созданием упомянутого инструментария.

4.3. 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, 15]. 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 по решению научно-исследовательских задач. Ее архитектура базируется на каркасе FP6 EC, нацеленном на инициативное 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. Этот набор будет расширяться путем добавления новых плагинов (plugins) [8] со специализированными услугами для конкретной области знаний. Каждый плагин включает публичный интерфейс с описанием услуг для потребителей или поставщиков. Первоначально этот 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 главным тормозом 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 скомпилированный код (например, при выполнении готовых 32-разрядных двоичных элементов на 64-разрядных платформах) сетевой среды Grid.

Следующая важная проблема – это 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. Сущность ее состоит в следующем. Модель данных, как и модель CIM, позволяет вводить формальные сущности в структуре проектов ПО, описывать объекты и взаимоотношения между ними, а также предоставлять результаты выполнения ПО. Внутреннее хранение данных основывается на модели данных реляционного типа, реализованной средствами MySQL [23]. Она поддерживает различные модели развертывания (Веб-сервис, Веб-приложение и БД), которые находятся на одном узле, а используемый сервис, база данных – на другом узле.

Описание модели данных основано на следующих базовых положениях:

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

– объект конфигурации содержит информацию о версии, связи с репозитарием, GUID, виде платформы и связи этого идентификатора с глобальным в конфигурации;

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

– при определении конфигурации и платформы в каждом объекте объявляется GUID, его свойства, среда выполнения и зависимости, которые могут быть объявлены статически или динамически. Статическая зависимость – это взаимоотношение между двумя конфигурациями, динамическая зависимость – взаимоотношение между конфигурацией и модулем.

Для реализации этих соглашений предлагается разработать специальные средства автоматизации (рис. 3), суть которых заключается в следующем:

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 типам данных (простым, структурным и сложным) языков программирования, с помощью которых будет генерироваться необходимая промежуточная среда взаимодействия разноязыковых компонентов, подсистем и проектов (рис. 3);

3) анализ и разработка системы управления автоматизированной генерацией для каждой взаимодействующей пары элементов проекта, нового элемент 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 [5], АПРОП [6].

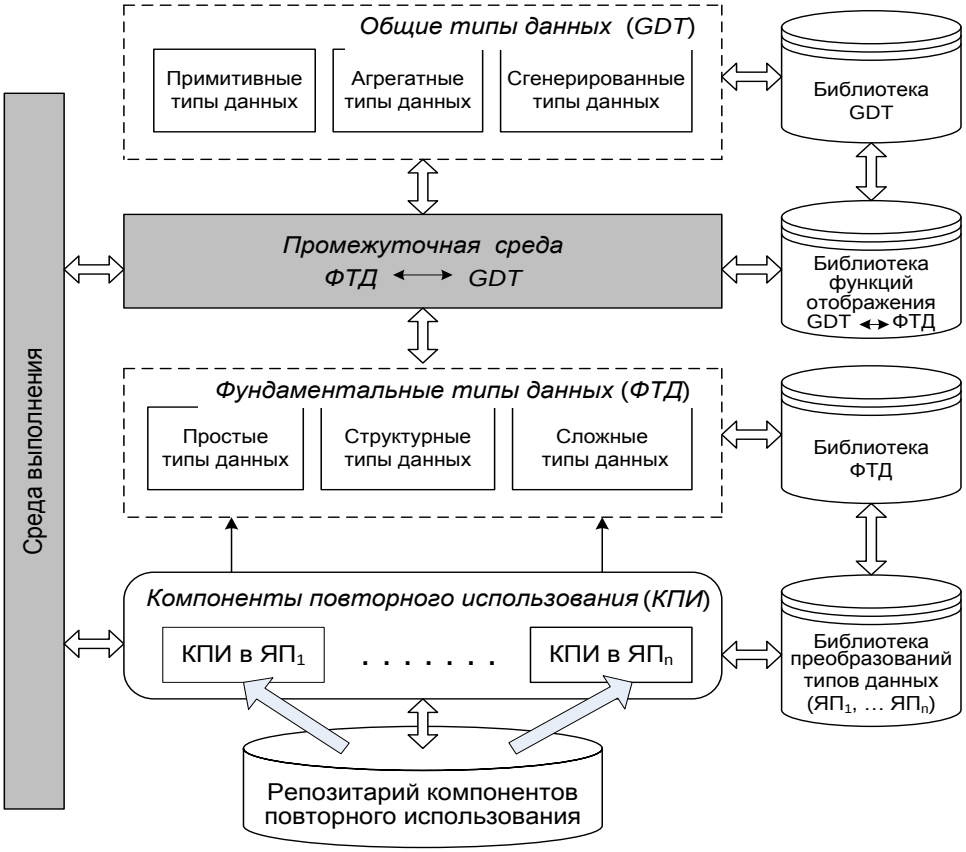

Рис. 3. Схема обработки типов данных GDT и ФТД

Таким образом, постоянно возникающую проблему интероперабельности разнородных компонентов из-за появления новых ЯП, архитектур платформ и сред можно решать путем построения программного инструментария на основе общих типов данных GDT, который будет адекватным появлению новых видов разнородных программных объектов.

Заключение

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

Источник реализационной точки зрения проблематики обеспечения взаимодействия разноязыковых программных объектов – анализ ныне действующих системных сред (COM, CORBA, SUN, MS.NET, JAVA, Grid и др.). Исследованы возможности названных сред, общность в реализации названной проблемы с помощью промежуточной среды (middleware), брокера запросов в распределенной среде и генерации необходимых промежуточных компонентов (stub, skeleton) в системе CORBA, DCOM, SUN, и унификации выходного программного кода в системе MS.NET для круга ее ЯП.

В рамках проекта по генерирующему программированию сформулирована новая концепция, отличающаяся от существующих тем, что она основана на методе генерации общих типов данных стандарта ISO/IEC 11404 применительно к новым ЯП, программы в которых будут обрабатываться в будущих гетерогенных средах. Проводится разработка варианта инструментальной поддержки системы обеспечения взаимодействия разноязыковых и разноплатформенных программ в среде ГП для допустимых в ней ЯП и сформулированы пути подключения новых ЯП для среды Grid. С участием студентов КНУ имени Тараса Шевченко будет создана библиотека функций преобразования GDT типов данных к соответствующим фундаментальным типам данных ЯП, библиотека функций для преобразования несоответствий их реализации системами программирования, отличиями в платформе компьютеров в форматах данных и разрядностью (32, 64) их памяти. Предлагаемая система является перспективной, она может быть реализована также для проекта Grid и в будущем взаимодействие компонентов, систем и сред будет стандартным и формализованным.

1. http://ru.wikipedia.org/wiki/
2. *Пройдиков Є.І., Теплицький Л.А.*Тлумачний словник з обчислювальної техніки, Інтернету і програмування. – К., 2006.– 833 с.
3. *Чернецки К., Айзенекер У.* Порождающее программирование. Методы, инструменты, применение.– Изд. дом Питер. – М.: – СПб. – Харьков. – Минск. – 2005. – 730 с.
4. *Электронная* наука. Состояние проблемы // ІПС НАН України. – 2007. – 94 с.
5. *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.
6. *Нейгел К., Ивьен Б., Глинн Д., Уотсон К., Скиннер М.* C# 2008 и платформа .NET 3.5 для профессионалов.: Пер. с англ. – М.: ООО Изд. Дом Вильямс, 2009. – 1392 с.
7. *Лаврищева Е.М., Грищенко В.Н.* Сборочное программирование. Основы индустрии программных продуктов: 2-изд. Допол. и перераб.– Киев: Наук. думка, 2009. – 370 с.
8. *Hoar K.* О Структурной организации данных // Структурное программирование.– М.: Мир, 1975.– С. 92 – 197.
9. *Турский В*. Методология программирования. Пер.с англ. – М.: Мир. – 1981.–265 с.
10. *Агафонов В.Н.* Типы и абстракция данных в языках программирования // Данные в языках программирования. – М.: Мир, 1982. – С. 267—327.
11. *Замулин А.В.* Типы данных в языках программирования и базах данных. – М.: Наука, 1987. – 152 с.
12. *Лаврищева Е.М.* Интерфейс в программировании // Проблеми програмування.– 2007.– № 2. – С. 126 –139.
13. *Лаврiщева К.М.* Генерувальне програмування програмних систем і сімейств // Проблеми програмування. – 2009.– № 1.– С. 3 – 16.
14. *Di Meglio A., Bégin M.E., Couvares P., Ronchieri E., Takacs E.* ETICS: the International Software Engineering Service for the Grid – Jornal jf PhysicsConference Series 119. – 2008.
15. *Castelli D., Candela L., Pagano P., Simi M.* 2005 2nd IEEE – CS International Symposium Global Data Interoperability (IEEE Computer Society) P. 6 – 99.
16. *Лаврiщева К.М.* Програмна інженерія. Підручник. – К.: Академперіодика, 2008.– 319 с.
17. *ДСТУ ISO/IEC TR 10000-2:*2004 Інформаційні технології. Основи та таксономія міжнародних стандартизованих профілів. Частина 2. Принципи та таксономія OSI профілів (ISO/IEC TR 10000-2:1998, IDT).
18. *Барлет Н., Лесли А., Симкин С.* Программирование на Java, Путеводитель.– К., 1996. – 736 с.
19. *Лаврищева Е.М.* Сборочное программирование. Теория и практика // Кибернетика и системный анализ. – 2009. – № 6. – С. 1 – 12.
20. *Бей И.* Взаимодействие разноязыковых программ.– М.– С. – Петербург. – Киев: Изд. дом «Вильямс», 2005. – 868 с.
21. *Грищенко В.Н.* Метод объектно-компонентного проектирования програмvных систем // Проблеми програмування. – 2007.– № 2. – С. 113 – 125.
22. *Лаврищева Е.М.* Методы программирования. Теория, инженерия, практика. – К.: Наук. думка, 2006. – 451 с.
23. *Шелдон Р., Мойе Д.* MySQL: базовый курс = Beginning MySQL. – М.: «Диалектика», 2007. — С. 880.
24. *Лаврищева Е.М.* Становление и развитие модульно-компонентной инженерии программирования в Украине. – Киев, 2008. – 33 с. – (Препр. Института кибернетики им. В.М.Глушкова НАН Украины; 2008-1).
25. *Фомичев В.С., Пютчлер У.* Промежуточные языки систем программирования // ЭВМ в проектировании и производстве. – Л., 1987. – Вып. 3. – С. 251 – 270.