Теория объектно-компонентного моделирования программных систем [1]

Е.М. Лаврищева <lavr@ispras.ru>
ИСП РАН, 109004, Россия, г. Москва, ул. А. Солженицына, дом 25
Московский физико-технический институт (государственный университет), 141700, Московская область, г. Долгопрудный, Институтский пер., 9

Аннотация. Представлен формальный аппарат объектно-компонентного моделирования предметной области, включающий логико-математическое описание на четырех уровнях проектирования объектной модели. На каждом уровне моделируются и уточняются объекты, устанавливаются их связи (интерфейсы) и отношения, базовые характеристики функций и логика поведения объектов. Объекты и их интерфейсы группируются в классы и отображаются в графе объектной модели. Объекты переводятся к виду компонентов с сохранением связей. Предложены формальные модели (компонента, интерфейса, среды и системы) компонентной среды, обеспечивается технический перенос объектов в компонентную среду и сборка объектов и компонентов в конфигурационный файл системы для выполнения.

Ключевые слова: моделирование, логико-математический аппарат, предметная область, уровни моделирования, модели, объекты, граф, компоненты, изменяемость, многоразовые компоненты, сборка, конфигурация.

Вступление

Построение модели предметной области (домена) получило название концептуального моделирования. В нем каждая модель отображает присущую предметной области (ПрО) систему понятий, характерные их свойства, отношения друг с другом и правила поведения. Модель домена ориентирована на понимание ее семантики человеком. Она содержит существенные признаки понятий объектов (денотаты, концепты), их размер, общие характеристики, их зависимости и содержание.

К средствам моделирования ПрО относятся: онтологические модели OWL (Web Ontology Language); ODM (Ontology Definition Metamodel); модели предметных областей – GDM (Generative Domain Model), Feature Model (Модель характеристик), MDD (Model Driven Development), MDA (Model Driven Architecture), SOA (Service-Oriented Model); UML1, UML2 и др. Кроме того используются языки RCL, Z, B, VDM и др. для формальной спецификации ПрО и доказательства правильности их описания.

[1] Работа поддержана грантом Российского фонда фундаментальных исследований № 16-01-00352.

В работе предлагается объектно-компонентный метод (ОКМ) [1-5], который обеспечивает логико-математическое моделирование системы на четырех уровнях (обобщенном, структурном, характеристическом и поведенческом). Каждый уровень детализирует объекты и формирует объектную модель (ОМ) системы в виде объектного графа и характеристической модели MF (Model Feature). Функции объектов преобразуются к виду компонентов с изоморфным отображением данных к виду форматов платформы взаимодействия объектов. Функции многоразового использования представляются компонентами повторного использования (КПИ), которые размещаются в репозитории готовых артефактов (reuses, component, services, assets и др.).

1. Теория моделировния систем из объектов предметной области

На рубеже 80–х XX столетия Гради Буч предложил объектно-ориентированный подход (ООП), который ориентирован на изменение используемого традиционного подхода к программированию больших систем, приведшего к *кризису сложности* создаваемых программных систем (ПС).

ООП был толчком к созданию новых объектно-ориентированных технологий, языков программирования (ЯП), библиотек классов объектов и примитивных функций преобразования типов данных (ТД) ЯП к формату платформы, а также инструментов поддержки ООП (COM, CORBA, DCE RPC, Rational Rose, UML и т.п.) в период 1992 –2000. Многие программисты перешли на новые инструменты ООП и испытали их вдоль и поперек. Таким образом, принципы ООП (наследование, полиморфизм, классы и др.) утвердились в программировании и используются в ЯП; UML2, C++, C#, Java, Ruby и др. Ежегодно выходит сборник трудов «Объектные системы» (X Международная конференция), в которых освещаются вопросы применения ООП, UML, CORBA в разных отечественных организациях.

В момент расцвета объектных систем нами сформирована теория моделирования систем из объектов – метод ОКМ и трансформации их к программным компонентам. Основу ОКМ составляет четырехуровневое моделирование объектных систем с помощью логико-математических операций. На первом уровне проводится декомпозиция предметной области на объекты. На следующих уровнях определяются их внешние характеристики MF и формируется объектный граф G, в вершинах которого находятся объекты, а дуги задают связи (интерфейсы) и данные, которыми обмениваемыми с другими объектами. На поведенческом уровне выявляется поведение объектов и строится модель поведения. Объекты графа переводятся к компонентам и переносятся в компонентный граф. Объекты и компоненты накапливаются в репозитории и могут конфигурироваться в разные варианты ПС. Метод ОКМ представлен в ряде публикаций [1-15], прошел апробацию

на курсах лекций «Программная инженерия» МФТИ (2001-2015) и в системе информатизации Украины.

Далее дается описание теоретических и реализационных аспектов ОКМ применительно к ПС и их семействам (СПС).

1.1 Математическое моделирование объектной модели

Математическая теория объектного моделирования ПрО построена с использованием детонационной теории Фреге и базовых понятий ООП Г. Буча:

- Класс – совокупности объектов с общими свойствами.
- Метод (или функция) – операция над экземплярами объектов класса.
- Наследование – сохранение атрибутов и операций родительского объекта класса.
- Инкапсуляция – доступность к методам объекта и упрятывание внутренней информации для изоляции особенностей реализации.
- Полиморфизм – возможность оперировать с объектами без ограничений.

Объект выделяется в процессе анализа ПрО и задается логико-математическими формализмами, функциями объектов и их характеристик в моделях OM и MF. Определяются взаимоотношения объектов, их поведение и классы согласно ООП.

Теория Г.Буча позволяет определять стратегию проектирования предметной области, исходя из утверждения, что весь материальный мир состоит из объектов. Любая предметная область – это совокупность объектов, связанных между собой некоторым множеством отношений (наследования) и поведения в течение заданного времени. То есть

< объектная ориентация> = <объекты> + <наследование>.

Каждое понятие ПрО, вместе с его свойствами и особенностями поведения является отдельным объектом, а вся ПрО – это совокупность объектов со связями, которые устанавливаются на основе отношений между этими объектами. В качестве объекта выступают как абстрактные образы, так и конкретные физические предметы или группы предметов с общими характеристиками и функциями.

При определении объекта используется понятийная структура – треугольник Фреге (рис. 1), согласно которого объект есть *денотат*. Математический символ (знак, слово, имя) используется для идентификации объекта (вершина **I**), а денотат (**II)** соответствует уровню знаний о сущности моделируемого мира, отражаемой этим объектом. Естественно, что денотат можно идентифицировать различным образом, используя для этого выбранные алфавиты, и одному объекту могут соответствовать несколько концептов (**III)**, отражая выбранный уровень абстракции. Каждому треугольнику Фреге в выбранной логической системе будет соответствовать определенный объект с

собственным именем, концепт которого задает семантическое содержание [10-13, 15].

Рис. 1. Треугольник Фреге для определения объектов

В соответствии с треугольником Фреге, объект состоит из собственного идентификатора, денотата – образа предмета или абстрактного компонента, на который указывает этот идентификатор, и концепта, задающего смысл, содержание денотата.

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

Свойство объекта определяется на множестве объектов ПрО унарным предикатом, который получает значение истины при наличии внешних и внутренних характеристик объекта.

Характеристика – это свойство, определенное на множестве внешних и внутренних характеристик с условием получения истины из области значений свойств.

Отношение определяется бинарным предикатом на множестве объектов ПрО, принимающих значение истины на заданной паре отношений объектов типа *IS –A / Part – Of*.

1.1.1 Изменение денотатов и концептов объектов

Все элементы треугольника Фреге – имя, денотат и концепт могут изменяться. Денотат, как некоторая сущность некоторой реальности, соответствует определенному объекту и может представляться в виде совокупности однородных или неоднородных элементов (предметов).

Эти совокупности могут изменяться д*екомпозиционным* или *композиционным методом*.

Декомпозиционные изменения концептов включают концепты новых детализированных объектов с учетом концепта объекта или без него.

Изменения концептов, которые соответствуют композиционному подходу формируются на основе анализа одинаковых концептов или отличающихся. При изменении уровня детализации (абстракции) концептов объекта исключается одно или несколько свойств или характеристик формируемого концепта для нового объекта с одинаковым денотатом. Каждая из операций изменения объекта имеет определенный приоритет и арность, а также связана с соответствующими допустимыми изменениями денотатов и концептов.

Каждый объект как сущность принадлежит некоторому множеству объектов $O= (O_0, O_1 ,..., O_n)$, где $O_i = O_i (Name_i, Den_i, Con_i)$, а $Name_i, Den_i, Con_i$ соответственно означают – знак (имя), денотат и концепт объекта. Поведение концепта $Con_i = (P_{i1}, P_{i2} ,..., P_{is})$ определяется на множестве предикатов $P=(P_1, P_2 ,..., P_r)$.

К базовым операциям изменения денотат относятся следующие.

Декомпозиционное изменение денотата:

Формирование новых денотат однородных объектов

$decds(O_i): O_i \rightarrow \{O_{i1} ,..., O_{ik}\}$,

где $O_i = O_i (Name_i, Den_i Con_i) \forall j\ Con_{ij} = Con_i;$

$Den_i = Den_{i1} \cup ... \cup Den_{ik}.$

Формирование нового денотата неоднородных объектов:

$decdn(O_i): O_i \rightarrow \{O_{i1} ,..., O_{ik}\}$,

где $O_i = O_i (Name_i, Den_i Con_i)\ \forall j\ Con_{ij} = \varnothing;$

$Den_i = Den_{i1} \cup ... \cup Den_{ik}.$

Композиционное изменение денотат

Композиция однородных денотат объектов включает:

$comds(O_{i1}, ..., O_{ik}): \{O_{i1} ,..., O_{ik}\}, \rightarrow O_i,$

где $O_i = O_i (Name_i, Den_i Con_i)$

$\forall j\ Con_i = Con_{ij}; Den_{i1} \cup ... \cup Den_{ik} = Den_i.$

Композиция неоднородных денотат объектов имеет вид:

$comdn(O_{i1}, ...,O_{ik}): \{O_{i1} ,..., O_{ik}\}, \rightarrow O_i,$

где $O_i = O_i (Name_i, Den_i Con_i)\ \forall j\ Con_i = \varnothing;$

$Den_{i1} \cup ... \cup Den_{ik} = Den_i.$

Расширение концепта объекта. Если $P_t \in P$, $P_t \in Con_i$ $P_t(O_i)$ принимают значение инстны, то $conexp(O_i, P_t): O_i \rightarrow O_i,$

где $O_i` = O_i`(Name_i, Den_i, Con_i`), Con_{ij} \cup \{ P_t \} = Con_i.$

Сужение концепта объекта. Если $P_t \in Con_i$, то $connar(O_i, P_t): O_i \rightarrow O_i` `$

где $O_i` = O_i`(Name_i, Den_i, Con_i`), Con_{ij} = Con_{ij} \setminus P_t.$

Аксиома 1. Если $P_1 \in P$, $P_t \in Con_i$ $P_t(O_i)$ принимает значение истины, то $conexp(O_i, P_t)$:

$O_i \rightarrow O_i$, где $Con_{ij} \cup \{P_t\} = Con_i.$

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

$$\forall\ i\ [(i>0)\ \&\ (O_i \in O_0)]. \qquad (1.1)$$

Между определенными элементами множества O существуют отношения принадлежности. На этом уровне выделяют сущности ПрО путем собственного субъективного восприятия и описывают их объектами в виде базовых понятий ОМ. Выделение сущностей осуществляют с учетом отличий, определяющих соответствующие им понятийные структуры, исходя из треугольника Фреге, и фиксируют объекты идентификаторами и концептами. Результат этого уровня анализа – ОМ. В ней на данном уровне размещены только имена объектов ПрО. Объект имеет хотя бы одно свойство или характеристику и уникальную идентификацию во множестве объектов, предикатов свойств и отношений между ними. Отношение определяется бинарным предикатом на множестве объектов ПрО, принимающих значение истины на паре *IS –A/Part – Of*.

Для проектирования ОМ используется логико-математический аппарат, применяемый на четырех уровнях.

1.1.2 Уровни логико-математического моделирования ПрО

Четырехуровневое объектное моделирование ПрО является развитием модульной парадигмы декомпозиции ПрО из функциональных и интерфейсных объектов и их определения и уточнения на I–IV уровнях логико-математического аппарата (рис. 2.):

I. **Обобщающий уровень** определяет базовые понятия ПрО – объекты без учета их сущности и свойств. Объект задается в виде денотата в соответствии с теорией Фреге в виде <имя объекта> <концепт>.

II. **Структурный уровень** определяет расположение объектов в структуре модели ПрО, устанавливает упорядочение объектов и представление их структуры в виде графа с операциями над объектами (объединения, пересечения, разности, приложения, симметричной разницы и др.

III. **Характеристический уровень** задает общие, специфические свойства и характеристики концептов объектов в логико-алгебраическом представлении объектов в виде графа с характеристиками и свойствами в модели МХ;.

IV. **Поведенческий уровень** определяет поведение и изменение объектов в зависимости от событий, которые они создают при их выполнении.

При выявлении объектов на обобщающем уровне, проводится их конкретизация к виду:

- класс, как объект, который представляет собой множество;
- экземпляр класса, как объект, который является элементом определенного множества, который сам является классом;
- объединенный класс, как множество, которое является прямой суммой нескольких других множеств;

- класс-пересечение, как множество, которое является общей частью других множеств;
- агрегированного класса, как множества, которое является подмножеством декартова произведения нескольких других множеств.

Логико-алгебраическая концепция структурного уровня проектирования рассматривает множество объектов ПрО как алгебраическую систему из совокупности объектов и предикатов определенной сигнатуры, которые соответствуют условиям концептуального моделирования объектов ПрО с помощью операций:

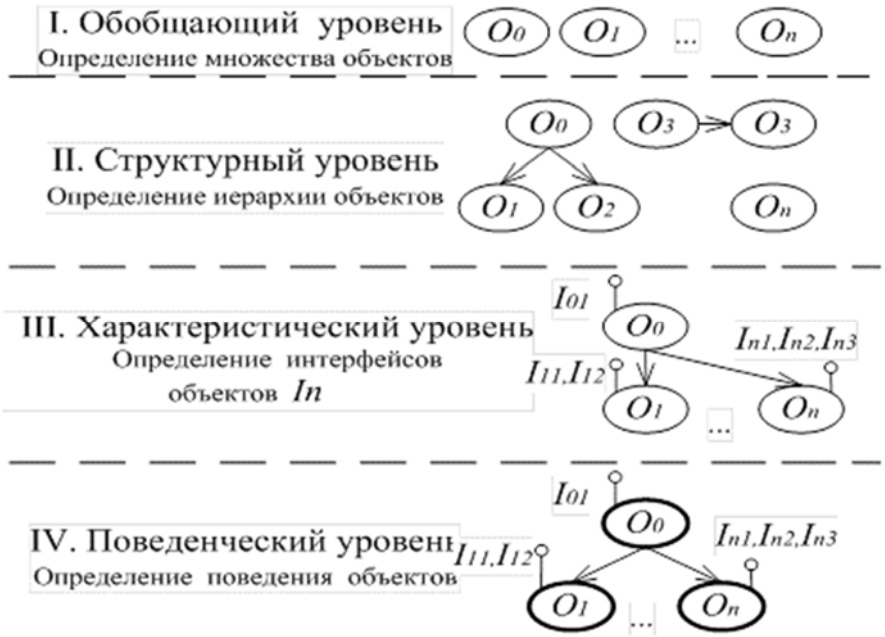


Рис.2 – Уровни проектирования ОМ

- 0-арной для задания констант в устоявшихся характеристиках ПрО;
- унарной для отображения свойств отдельных объектов;
- бинарной для задания взаимосвязей между отдельными парами объектов.

Объект, имеющий одновременно статус множества (класса) и элемента какого-либо множества, имеет внешние и внутренние свойства и характеристики.

Характеристика объекта – это совокупность свойств (унарных предикатов) и которое является подмножеством множества, выделенных в системе предикатов, удовлетворяя условию, при которой каждый объект принимает

значение истины одновременно не больше, чем один предикат из совокупности. Характеристика может быть внешней и внутренней.

Внешняя характеристика предназначена для описания проблемной ориентации совокупности объектов и их статуса как элементов множеств. Каждое внешнее свойство входит в состав объектной характеристики одного проблемного аспекта ПрО.

Внутренняя характеристика предназначена для определения статуса объекта, как одного из критериев формирования множества из объектов, эквивалентного внешним свойствам.

Внешние и внутренние характеристики объектов задаются списком или множеством свойств с разными типами. При этом, список свойств любой из этих характеристик содержится только в соответствующей модельной характеристике $MF = (MF_1, MF_2, \ldots, MF_n)$.

К операциям модификации взаимоотношений объектов относятся:

- экземпляризация объекта-множества с заданием различных внешних свойств, принадлежащих внутренней характеристике класса;
- классификация объектов, которые группируются по внешним свойствам.

Свойства, соответствующие одной и той же характеристике, могут использоваться в концепции структурной упорядоченности при конкретизации объектов и определении их принадлежности к классу. Определение принадлежности объектов достигается путем реализации отношение принадлежности типа "Part-of", операции детализации и агрегации.

Рассмотрим более подробно уровни проектирования модели ПрО.

Обобщающий уровень проектирования ОМ

Этот уровень задает наивысшую меру абстракции отображения ПрО в ОМ с использованием концепции обобщения с целью определения сущностей ПрО и представление их в виде объектов ОМ как базовых понятий модели. Результатом является спецификация объектов вида:

<имя объекта> <концепт>,

где *имя объекта* – идентификатор из символьной строки литер и десятичных цифр;

концепт – текст, который определяет семантику <денотат объекта>.

Аналитик выделяет сущности ПрО путем субъективного восприятия последней и описывает их в виде объектов как базовых понятий проектируемой модели. Выделение сущностей осуществляется с учетом изменений, которые определяются понятийными структурами – треугольниками Фреге и фиксируется идентификаторами и концептами. Результат этого проектирования – ОМ, в которой размещены именованные объекты ПрО, образующие множество $O=(O_0, O_1, \ldots, O_n)$, где O_0 – объект ПрО и множество $O'=(O_1, \ldots, O_n)$ денотатов и концептов объектов после декомпозиции и композиции.

Между определенными элементами множества O' существуют отношения принадлежности ($\in$). На этом уровне выделяют сущности ПрО путем субъективного восприятия и описания их в виде базовых понятий ОМ. Выделение сущностей осуществляют с учетом отличий, определяющих соответствующие им понятийные структуры, исходя из треугольника Фреге. Объекты получают идентификаторы и концепты. Результат этого уровня анализа – ОМ. В ней на данном уровне размещены только имена объектов ПрО и правила изменения денотатов.

Структурный уровень моделирования ОМ

Объекты уже определены на обобщающем уровне абстракции. Каждый объект задается как множество или элемент множества.

Пусть $O=(O_0, O_1, O_2 \ldots, O_n)$, исключая из множества O элемент O_0, который не принадлежит другим элементам, получаем множество $O'=(O_1, O_2, \ldots, O_n)$.

Выражение (1.1) на этом уровне трансформируют в такое:

$$\forall i \; \exists j \; [(i>0) \; \& \; (j>0) \; \& \; (i \neq j) \; \& \; (O_i \in O_j)]. \qquad (1.2)$$

В выражении (1.2) каждый из объектов является множеством или элементом определенного множества и к ним применяются операции теоретико–множественной алгебры. Это выражение определяет отношение часть–целое, экземпляризации и агрегации.

Каждый объект на структурном уровне представляется как множество или элемент множества. Пусть $O'=(O_1, O_2, \ldots O_n)$, $OS=(OS_1, OS_2, \ldots OS_m)$ i $OS \subseteq O'$.

Тогда IS: $O' \rightarrow OS$ есть ограничение тождественного отображения O' на себя. Если $S=(S_1, S_2, \ldots S_m)$ – множество объектов, которые определены на данном уровне, то TS: $OS \rightarrow S$, где каждому OS_i соответствует S_i. Т.е определено отображение между объектами на обобщающем и структурном уровнях $TS*IS$: $O' \rightarrow S$.

Пусть $\Omega = (\cup, \cap, /, \Diamond, \oplus, -)$ – совокупность теоретико-множественных операций. Тогда $\Sigma = (S, \Omega)$ определяет алгебраическую систему для структурно-упорядоченного уровня. Оно базируется на теоретико-множественной концепции, в соответствии с которой объекты, определенные на обобщающем уровне, предоставляются как множества или элементы определенного множества алгебраической системы $\Sigma = (O', \Omega)$ структурного уровня.

Сущность теоретико-множественных операций:

- *объединение* $\bigcup (A, B)$, где A и B – объекты со статусом множеств. Результат применения операции: новый объект-множество, которое получено объединением множеств A и B;
- *пересечение* $\cap (A, B)$, где A и B – объекты-множества. Результат применения операций: новый объект-множество, который является пересечением множеств A и B;

- *вычитание* $D = A \oplus B$ при $B \subset A$. Результат применения операции: новый объект-множество со всеми элементами A, которые не входят в B;
- *симметричное вычитание* $S = A \setminus B$, где $A \cap B$, и не $A \subset B$, не $B \subset A$. Результат применения операции – это новый объект-множество с множеством элементов, которые принадлежат A или ("или" – распределенное) B.

При применении таких операций к каждой паре объектов проводится выявление корректности структурной упорядоченности и определение новых объектов, которые не выявлены на обобщающем уровне.

Таким образом, на данном уровне осуществляет решение следующих задач:

- определение и фиксация структурной упорядоченности объектов;
- расширение описания объекта или конкретизацию объекта в структуре ПрО;
- определение новых объектов, связанных с заданными объектами;
- формирование объектного графа ПрО.

Принцип вычитания обеспечивает структурную упорядоченность объектов или системы путем применения отношения принадлежности. Выделение объекта и фиксацию его внешних различий осуществляется на основе анализа его структурного представления и определения места в объектном графе ПрО.

Результат проектирования на этом уровне – граф ОМ (рис.3) с установленными отношениями между объектами с учетом следующих требований:

- множество вершин графа задает взаимно однозначное отображение множества объектов в граф этого уровня, определенных на обобщенном уровне.;
- для каждой вершины существует хотя бы одна связь (структурная) с другой вершиной графа ($\rightarrow$) ;
- существует лишь одна вершина O_1 графа G (рис.3), которая имеет статус множества объектов, отображающего ПрО в целом.

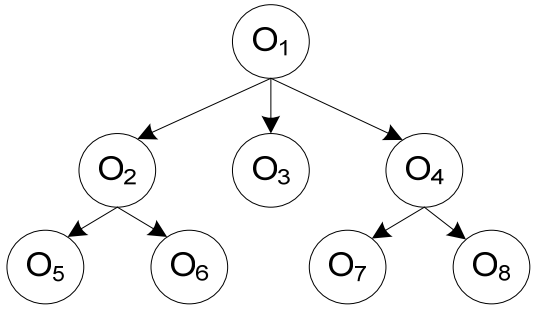


Рис. 3 – Структура графа G ={O}

Построенный граф G корректируется и реструктурируется с помощью операций объединения, пересечения, дополнения и др. Затем проводится контроль этого графа на структурную упорядоченность новых объектов, которые не были выявлены на обобщающем уровне.

Таким образом, при структурной упорядоченности граф расширяется следующими операциями:

- *конкретизация объектов*, при которой, возможно, единственный объект – элемент множества – обладает внешним свойством, эквивалентным одному из внутренних свойств объекта-множества;
- *принадлежность объектов* состоит в том, что элементы множества при условии владения одним и тем же внешним свойством (или совокупностью свойств) обладают внутренним свойством этого множества.

Свойства, которые соответствуют одной и тот же характеристике, могут использоваться в процессе структурной упорядоченности во время решения задач, как конкретизация объектов, так и определение их принадлежности.

При решении первой задачи они вносят критерий изменения объектов и поэтому могут быть разными для каждого объекта-элемента множества с конкретизирующими свойствами.

При решении второй задачи задается критерий принадлежности одинаковый для каждого объекта-элемента множества (свойства, которые определяют принадлежность). Частный случай отношения принадлежности – отношение *'Part–of'*,

Контроль полноты и не избыточности графа ОМ обеспечивается путем устранения продублированных объектов в разных множествах. Объектов системы.

При структурной упорядоченности граф определяются операции конкретизации и принадлежности объектов. Контроль полноты и не избыточности графа ОМ позволяет устранять сдублированные объекты в разных множествах.

Характеристический уровень проектирования ОМ

Объекты, которые определены на структурном уровне абстракции, на данном уровне с помощью логико-алгебраических операций позволяют провести модификацию объектного графа $G = \{O\}$ и конкретизацию их свойств и характеристик на основе предикатов множества P над множеством O.

Входная информация данного уровня – объектный граф G со спецификациями объектов. Результат уровня – откорректированный и расширенный граф ОМ новыми объектами с дополнительными характеристиками и свойствами. Для задания характеристик и свойств (предикатов) строится алгебра. На структурном уровне абстракции объектам задаются свойства и характеристики. Для них используются множество предикатов $P=(P_1, P_2 \ldots, P_r)$

на множестве S и $OA=(OA_1, OA_2 . OA_к)$, где для каждого OA_I существует P_j такой, что P_j на OA_i принимает значение истины.

Тогда $IA: S \rightarrow EA$ есть ограничение тождественного отображения S на себя.

Если $A=(A_1, A_2, \ldots A_k)$ – множество объектов, которые определены на данном уровне, и $TA: OA \rightarrow A$, где каждому OA_i соответствует A_i., то определяется отображение между объектами на структурном и характеристическом уровнях

$$TA * IA: S \rightarrow A.$$

$\Omega = (A, P)$ определяет модель алгебры характеристического уровня на множестве А и предикатов Р.

В соответствии с характеристическим аспектом для каждого из объектов формируется его концепт. Если $O'=(O_1, O_2, \ldots O_n)$ – совокупность объектов ПрО, а $P'=(P_1, P_2, \ldots P_r)$ – множество унарных предикатов, которые связаны со свойствами объектов ПрО, то концепт O' объекта O_i является множеством утверждений, которые построены на основе предикатов с P', принимающих значение истины для соответствующего объекта. То есть $Con_i = \{A_{ik}\}$ і $P_k(O_i) = true$ O. Согласно структур концептов между объектами определяются отношения типа «род-вид".

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

Аксиома 1. Каждый объект ПрО имеет хотя бы одну характеристику, которая задает семантику и уникальную идентификацию во множестве объектов ПрО.

Характеристики формируются аналитиком как множество свойств. Это множество может расширяться и модифицироваться в процессе формирования модели ОМ. Характеристики определяются типом объекта и принадлежат только одному объекту модели ПрО. Они предназначены для определения структурного статуса объектов (элементов или множеств), и подразделятся на внешние и внутренние характеристики, смысл которых определены выше.

Характеристики задаются в виде списка или множества свойств с несовпадающими типами, или соответствующие разным модельным характеристикам. Любой объект множества или элемент его может соответствовать только одной внешней и внутренней характеристики, то есть:

- объект-множество – внешняя и внутренняя характеристика;
- объект-элемент – внутренняя характеристика.

Основное условие определения принадлежности объекта к множеству эквивалентному внутреннему свойству объекта-множества.

На характеристическом уровне функциональным объектам определяются виды связи - интерфейс, в котором передаются внешние характеристики другим элементам, связанным на графе отношением принадлежности.

В связи с этим, к ранее построенному графу G добавляются характеристики модели MF и интерфейсы:

$$G = \{O, I, R\}, \qquad (1.3)$$

где O – множество объектов (функций),

I – множество интерфейсов,

R – множество отношений (relations) между объектами.

В граф G добавляются объекты I (рис.4), которые являются интерфейсными объектами, т.е. выполняют функцию вызова объектов и передачи им соответствующих данных в требуемом формате.

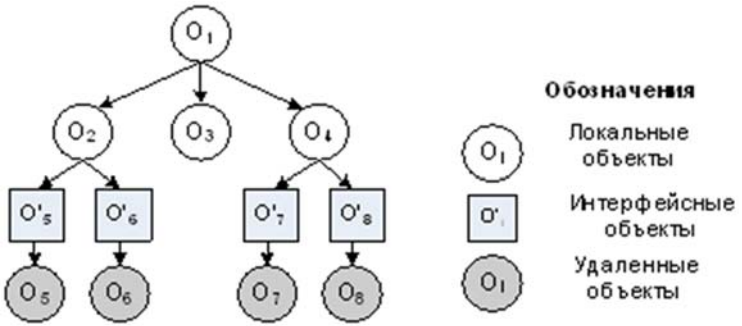


Рис. 4. Граф G на множестве объектов и интерфейсов

Вершины данного графа G залают функциональные объекты – O_1, O_2, O_3, O_4 , O_5, O_6 , O_7, O_8 и интерфейсные объекты – O'_5, O'_6, O'_7, O'_8, которые размещаются в репозитории, а дуги соответствуют отношениям между всеми видами объектов.

Элементы графа $O_1 – O_8$ описываются в ЯП, а интерфейсные объекты $O'_5 – O'_8$ в специальном языке интерфейса IDL (Interface Definition Language), разработанного в рамках проекта CORBA. Параметры внешних характеристик интерфейсных объектов передаются между объектами через интерфейсы и помечаются как *in* (входной), *out* (выходной), *inout* (входной и выходной) в IDL.

Функциональные объекты графа G соответствует методам реализации этих объектов ПрО. При конкретизации функциональные объекты графа G устанавливают связь с другими объектами через интерфейсные параметры *in*, *out*, *inout* множества I.

Интерфейсные объекты графа содержат описание передаваемых данных, операторы удаленных вызовов RPC или RMI и передачи данных. При необходимости передаваемые данные преобразуются брокером ORB к соответствующим форматам среды выполнения метода объекта.

По графу G (рис.4) можно построить программы $P_1 – P_5$ с использованием операторов объединения (сборки) *link*:

1) $P_1 = O_2 \cup O_5$, *link* $P_1 = In\ O'_5$ $(O_2 \cup O_5)$;
2) $P_2 = O_2 \cup O_6$, *link* $P_2 = In\ O'_6$ $(O_2 \cup O_8)$;
3) P_3 ;

4) $P_4 = O_4 \cup O_7$, *link* $P_4 = In\ O'_7$ $(O_4 \cup O_7)$;
5) $P_5 = O_4 \cup O_8$, *link* $P_4 = In\ O'_8$ $(O_4 \cup O_8)$;
6) $P_0 = (P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5)$.

Результат связи двух объектов (например, *link* $P_2(O'_6)$ - это интерфейсный объект $In\ O'_2$. В этом объекте задается список входных интерфейсов, совпадающих с множеством интерфейсов объекта–приемника, а также список выходных интерфейсов, которое совпадает с множеством выходных интерфейсов объекта–передатчика.

Аксиома 2. Расширенный граф G с интерфейсными объектами, структурно упорядочен (наверх), проконтролирован на полноту, избыточность и дублирующие элементы.

Объекты могут иметь несколько интерфейсов, которые могут наследовать интерфейсы других объектов в том случае, когда они предоставляют данные всему множеству входных интерфейсов.

Множество объектов и интерфейсов графа отмечаются общими характеристиками объектов в ОМ. Они считаются достоверными, если выполняется условие: каждая внутренняя характеристика эквивалентна внешней характеристике объекта. Если это условие не выполняется, то такой элемент удаляется из множества O и из графа соответственно.

Контроль объектного графа ОМ осуществляется с помощью операций логико-алгебраических операций, тем самым иллюстрируется общность идеологии подхода к созданию ОМ из объектов и выявление ситуаций нарушения структурной упорядоченности объектов путем анализа свойств и характеристик объектов множества. Эти операции обеспечивают группирование с помощью операций бинарных отношений типа *'is-a'* и для пары "объект-множество – объект-элемент" выполняются операции:

- э*кземпляризации* элемента путем сопоставления совокупности объектов, каждый из которых обладает внешними свойствами, принадлежащими внутренней характеристике объекта этого множества;
- *классификации* объектов согласно их внутренней характеристики, разные значения которой относятся к внешним свойствам объектов, которые классифицируются.

Паре "объект-множество – объект-множество" соответствуют операции:

- *обобщение* ситуации, при которой все внутренние свойства первого операнда являются подмножеством внешней характеристики второго операнда;
- *специализации (* как обратная операция по отношению к первой).

Для реализации бинарного отношения *'Part–of'* пары "объект-элемент – объект-элемент" используются операции:

- *агрегации* совокупности объектов и их внешних характеристик с указанием подмножество свойств порождения нового объект-множества с внутренней характеристикой из этого подмножества;
- *детализации*, которая является обратной по отношению к операции агрегации и позволяет определить список объектов, из которых состоит агрегированный объект-множество.

Поведенческий уровень проектирования ОМ

Входными данными для этого уровня являются ОМ на предыдущих уровнях. На данном уровне определяется последовательность состояний объектов и процессы перехода состояний. Взаимосвязи между объектами формируются с помощью бинарных предикатов, которые связаны со свойствами объектов ПрО, и детализируют взаимосвязи между состояниями объектов.

Данный уровень базируется на системе предикатов определяющих подмножество *SA* и множество объектов *S*, которые отображают сущность объектов ПрО и определяют их поведение, без учета внутренних свойств.

Совокупность предикатов задается через $P = (P_1, P_2 ..., P_r)$. Тогда, если поставить в соответствие множеству *SA* алгебраическую модель *A*, в которой *SA* является основным множеством, то на данном уровне абстракции объекты определяются как элементы множества *SA* с использованием предикатов *P*, и на этих элементах принимается значение "*истина*".

В общем случае этот уровень позволяет рассматривать зависимость от алгебры *A* (элементы множества *SA* и *P*) на характеристическом уровне, с отображением зависимости от времени *t* реальных объектов ОМ и их характеристик.

Пусть $B = (B_1, B_2 ..,. B_k)$ – множество объектов, которые определены на поведенческом уровне, и каждому A_i отвечает $B_i(A_i \rightarrow B_i(t))$. Тогда отображение ТВ определяется между объектами на характеристическом и поведенческом уровнях

$$TB: A \rightarrow B.$$

Определение состояния объектов зависит от конкретного значения параметра времени на отрезке часовой оси (отрезок отвечает длительности существования объекта) $SA_j = SA_j(t)$, в которой SA_j – объекты-элементы множества *SA* с ассоциируемой совокупностью предикатов *P*, которые принимают на этих объектах значение "*истина*".

Теоретически существует зависимость самих предикатов относительно времени, то есть зависимость вида $P_i = P_i(t)$. Однако для упрощения модели подобная зависимость в данной работе не учитывается, а возможные изменения времени рассматриваются как новые предикаты.

Понятие времени – это абстрактное понятие, оно соответствует конкретному параметру системы, значения которого упорядочены, и каждому из них соответствует состояния объектов, к которым относятся: атрибут состояния, статический и динамический атрибут состояния.

Динамический атрибут состояния зависит от значения параметров, моделирующих во времени процесса выполнения. Предлагается два варианта моделирования зависимости объекта от времени:

а) относительно времени в виде последовательности состояний системы, упорядоченных в соответствии с параметром, который задает время исходя из модели событий;

б) характеристики объектов, которые зависят от параметра времени в модели состояний.

ОМ задает модель состояний:

1) для каждого объекта последовательно рассматривается характеристика, которая ассоциируется с ним. В списке свойств, которые составляют множество значений характеристики, указывается либо статическое свойство, либо набор свойств, которые определяют динамику поведения объекта. Последние составляют подмножество значений характеристики, которое определяет динамический атрибут состояния объекта в зависимости от параметра времени. Статические или динамические свойства определяются для каждой характеристики объекта отдельно. Состояние объекта ОМ определяется совокупностью статических атрибутов (свойств) объекта и набором динамических атрибутов в каждый момент моделирования времени существования объекта;

2) переход объекта из одного состояния в другое осуществляется посредством некоторого сообщения (или события), которое является результатом анализа состояния ОМ (в случае ее предыдущего изменения). Переход порождает определенное событие, которое, в свою очередь, инициирует анализ состояния ОМ;

3) изменения состояния объекта выполняются посредством набора методов, ассоциируемых с этим объектом. Кроме того, объекту соответствуют не только виды сообщений (событий), порождающие изменение состояний и список сообщений, на которые объект реагируют, как на сообщения времени.

Переход объекта из одного состояния в другое осуществляется посредством изменения состояния объекта с помощью методов, ассоциируемых с этим объектом. Кроме того, объекту соответствуют не только виды сообщений (событий), порождающие изменение состояний, но и список сообщений, на которые объект реагируют, как на часовые сообщения.

С каждым состоянием объекта связано некоторая деятельность (*Activity*), которая дальше будет иметь название действия (*Action*), и происходить в тот момент, когда объект достигает определенного состояния. Операции действия имеют доступ к модели состояний и к данным других объектов.

Модель состояний может задаваться в виде модели переходов состояний. Некоторые модели состояний имеют одно или более состояний, где объект появляется в первый раз.

В некоторых моделях состояний одно (или, возможно, несколько) состояние является концом ЖЦ объекта, или заключительным состоянием и задает одну из двух ситуаций:

- объект становится неподвижным во времени, то есть продолжает существовать, и без динамического поведения;
- объект прекращает существование.

Если при моделировании поведенческого уровня ОМ возникает необходимость изменения объектов, их свойств или характеристик, отношений между объектами, то их результаты соответствуют коррекции модели поведенческого уровня.

Классификация объектов

Объекты ОМ могут группироваться в классы объектов. Если объект – элемент другого объекта, то он определяется классом.

Класс – это определенное множество объектов, имеющих общие переменные, структуру и поведение. Одним из основных свойств объектов является инкапсуляция. Она реализуется с помощью интерфейсов, которые состоят из методов и атрибутов. С общей точки зрения, интерфейс экземпляра объекта состоит из совокупности методов. Каждый объект может иметь несколько интерфейсов, которые определяют его функциональные свойства. Кроме того, объект может иметь специальный интерфейс, методы которого работают с целыми экземплярами (например, Home-интерфейс в модели EJB языка Java). В состав этих методов входят методы поиска экземпляров компонента, их создание, уничтожение и т.п. Например, поиск экземпляра объекта, компонента может происходить по его уникальному имени или значению определенной переменной.

Определение объекта формулируется согласно условия: каждый объект обязательно является множеством или элементом некоторого множества.

Упорядочение объектов ОМ выполняется с учетом отношения принадлежности, а элементов множества с помощью множества целых чисел.

Конкретизацией понятия объекта в данной концепции являются:

класс – это объект, который задает собой множество;

экземпляр класса – объект, который является элементом определенного множества, который сам является классом;

объединенный класс – множество, которое является прямой суммой нескольких других множеств;

класс–пересечение – это множество, которое является общей частью других множеств;

агрегированный класс – это множество, которое является подмножеством определенного декартового произведения нескольких других множеств.

Объекты объединяются в классы в соответствии с общими характеристиками. Модель ОМ с классами имеет вид:

$$OM = (Oclass, GK), \quad (1.5)$$

где $Oclass = \{Oclass_i\}$ – множество классов объектов функций или методов с общими свойствами; GK – объектный граф, который устанавливает связи и отношения между экземплярами класса.

Каждый класс представляется в виде:

$$Oclass_i = (ClassName_i, Meth_i, Field_i\}, \quad (1.6)$$

где $ClassName_i$ – имя класса, $Meth_i = \{Methj_i\}$ – множество методов, $Field_i = \{Field_n^i\}$ – множество переменных, которые определяют состояние экземпляров класса.

Пусть $Pfield^i \subset Field^i$ – множество внешних переменных (public), которые доступны вне. Каждому $Pfield_n^i \in Pfield^i$ і поставим в соответствие методы get <$Pfield_n^i$> и set <$Pfield_n^i$> для присвоения и выборки значений соответствующей переменной, то есть эти переменные становятся атрибутами в терминах современных компонентных моделей. Соответственно, в других классах вместо обращения к таким переменным будут использоваться указанные методы. Тогда множество методов имеет вид:

$Imethod_i = Method_i^i \cup \{get<Pfield_n^i>\} \cup \{set<Pfield_n^i>\}$, которому сопоставляется интерфейс $Ifunc_i$, состоящий из прототипов методов, входящих в $Imethod_i$.

Параллельно с *Osyst* рассматривается система *Isyst* = (*Ifunc, IG*),

где *Ifunc* = {$Ifunc_i$} – множество интерфейсов;

IG – интерфейсный граф, идентичный графу G.

Класс $Oclass_i$ порождает свои экземпляры (объекты) $Obj_k^i = \{ObjName_k^i, Method^i, Field^i\}$, которым в системе *Isyst* будут соответствовать интерфейсные элементы $Iobj_k^i = \{Iname_k^i, Ifunc_i\}$.

Для каждого такого элемента не определена реализация соответствующего интерфейса. Сопоставив некоторому интерфейсу реализацию $ImpFunc^j$, то есть сформируем интерфейсный элемент

$Iobj_k^{ij}$ = {$Iname_k^i$, $Ifunc^i$, $ImpFunc^j$ }, который, по своей сути, эквивалентный экземпляру объекта $Cins_k^{ij} = (Iins_k^{ij}, IntFunc^i, ImpFunc^j)$.

В процессе своего функционирования объект с помощью метода Create интерфейса *Cfact* порождает экземпляры: $Cfact.Create: Comp \rightarrow \{Cins_k^{ij}\}$,

$$Cins_k^i = (Iins_k^i, IntFunc_i, ImpFunc_i),$$

где $Cins_k^j$ – экземпляр компонента с уникальным именем $Iins_k^i$, который предоставляет свою функциональность с помощью интерфейса $IntFunc^i$ и реализацию этого интерфейса – $ImpFunc_i$.

Модель ОМ в конкретной распределенной среде выполняется с помощью сообщений, передаваемых на основе графа G и через интерфейсную вершину

к удаленному объекту и обратно. Такая схема взаимодействия поддерживается многими современными распределенными средами, например, система Corba, где брокер посылает сообщения с интерфейсными данными некоторому объекту для выполнения и возвращения результатов.

1.2. Формальные основы объектного анализа

Принципы объектного анализа

Принцип всеобщности означает, что на произвольном шаге объектного анализа все сущности – суть объекты.

Аксиома 2. Предметная область, которая моделируется из объектов, сама является объектом.

Аксиома 3. Моделируемая предметная область может быть отдельным объектом в составе другой предметной области.

Принцип существенности объектных различий. На произвольном шаге объектного анализа каждый объект является уникальным элементом.

Аксиома 4. Каждый объект имеет по крайней мере одно свойство или характеристику, которая задает его уникальную идентификацию во множестве объектов.

Принцип объектной упорядоченности. На произвольном шаге объектного анализа все объекты упорядочены в соответствии с отношениями между объектами.

Аксиома 5. Каждый объект имеет при необходимости одно отношение с другим объектом, которое обеспечивает его упорядоченность в рамках этой пары объектов.

Принцип целостности объектной модели. На произвольном шаге объектного анализа совокупность объектов и отношений между ними однозначно определяют объектную модель предметной области для определенного уровня ее абстракции.

Аксиома 6. На произвольном шаге объектного анализа объектную модель можно представить в виде ориентированного связного графа, вершинами которого являются объекты, а дугам соответствуют отношения между объектами.

1.2.1. Алгебра объектного анализа

Главной особенностью уровней проектирования является определение внешних и внутренних характеристик, которые наследуются в суперкласс и классы по схеме:

<объектная ориентация> = <объекты> + <отношения> + <наследование>.

В качестве объектов выступают как абстрактные образы, так и конкретные физические предметы или группы предметов с указанным подмножеством их характеристик и функций.

Алгебра объектного анализа ПрО это:

$$\Sigma = (O', I', A'), \qquad (1.7)$$

где $O'=(O_1, O_2 \dots O_n)$ – множество функциональных объектов,

$I = (I_1, I_2 \dots, In)$ – множество интерфейсов;

$A' =(A_1, A_2 \dots, A_n)$ – множество (Action – *A'*) операций над элементами множества *O*.

Каждая из операций *A'* имеет приоритет и арность и задает связь с соответствующими концептами объектов и операциями множества *A' = {decds, decdn, comds, comdn, conexp, connar}*, где *decds, decdn* – декомпозиции, *comds, comdn* – композиции и *conexp, connar* – сужение.

Семантики взаимодействия объектов графа *G*, функционирующего в среде «клиент – сервер», содержит:

O – множество функциональных объектов $O=O_1, O_2 \dots, O_k$;

I – множество интерфейсных объектов *In, Out,* в котором *In* – множество входных интерфейсных объектов, где задаются данные клиента для передаче их серверному объекту $O_k \in O, In(O_k)$ и *Out* – множество выходных интерфейсных объектов сервера $O_k \in O$, *Out* (O_k) и *Inout* – промежуточные интерфейсы.

Модель взаимодействия объектов основывается на свойствах и характеристиках объектов модели ОМ, которые представлены интерфейсами I в языке IDL (для задания параметров In, Out) и операциями принадлежности:

$O_k \in O, In(O_k)$ – множество входных *(In)* интерфейсных объектов клиента;

$O_k \in O, Out(O_k)$ – множество выходных (Out) интерфейсных объектов сервера.

Результатом взаимодействия двух объектов будет объект, в котором множество входных интерфейсов совпадает с множеством выходных интерфейсов объекта-сервера, а множество выходных интерфейсов – с множеством входных интерфейсов объекта-клиента:

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

Взаимодействие объектов $O_k \cdot O_l$ является корректным, если объект-сервер полностью обеспечивает сервис, необходимый объекту-клиенту, то есть

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

Удаленные объекты могут иметь несколько интерфейсов, они могут унаследовать интерфейсы других объектов ($O_k \leftarrow O_l$). Тогда последние предоставляют сервис всего множества исходных интерфейсов: $O_k \leftarrow O_l \Rightarrow Out(O_k) \subseteq Out(O_l)$.

В случае, когда объект унаследовал другой объект, в котором множество выходных интерфейсов содержит все его интерфейсы, а множество входных интерфейсов содержит только интерфейсы, необходимые для предоставления сервиса, то имеем

$$O_k \leftarrow O_l = \begin{pmatrix} Out(O_k) \cup Out(O_l), \\ \{I_m : (I_m \in In(O_k) \cup In(O_l)) \wedge \exists I_n \in Out(O_k \leftarrow O_l) : exec(I_n, I_m)\} \end{pmatrix}.$$

Над объектами и интерфейсами выполняются операции:

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

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

– проекция о*бъекта с* взаимодействующим объектом, которое задается равенством:

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

Унаследованный объект делегирует все интерфейсы и имеет свойства:

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

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

Операция параллельного выполнения программ ПС в распределенной среде не всегда является симметричной:

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

Результатом отображения объекта на объект через интерфейс осуществляется с помощью множества входных интерфейсов $O_k[O_l] = O_k[In(O_l)]$ или выходных интерфейсов $O_k[O_l] = Ok[Out(O_i)]$.

Модель ОМ в конкретной распределенной среде выполняется с помощью сообщений, передаваемых на основе графа G в сеть от объекта через интерфейсную вершину к удаленному объекту и обратно. Такая схема взаимодействия поддерживается многими современными распределенными средами.

Теорема 1.1. Множество операций Σ алгебры является полной системой операций и предикатов *P* относительно функций объектного анализа ПрО.

1.2.2. Определение моделей ПрО и ПС при моделировании ПрО

Элементами модельной среды являются: модель ПрО, объектная, компонентная модель ПС и модель характеристик артефактов и системы.

Объектная модель проектируется на уровнях и имеет вид:

$$OM = \langle G^t_1; G^t_2, G^t_3, G^t_4 \rangle, \quad (1.8)$$

где G_1 – граф объектов ПрО на обобщающем уровне ее проектирования (t = 1);

G_2 – FM граф структурного уровня (t=2);

G_3 – граф характеристического уровня с формированием моделей (FM, ОМ и КМ) (t = 3);

G_4 – модель взаимодействия объектов на поведенческом уровне (t = 4).

Объектам функций G^t_1 и их характеристикам соответствуют методы и данные, определяемые на уровнях 2, 3, необходимые для их реализации в ПС и обеспечения их взаимодействия

Компонентная модель ПС – развитие ОМ, в которой методы объектов представлены программными компонентами или КПИ и интерфейсами взаимодействия между ними. Модель СМ следующий вид:

$$CM = \langle RC, In, ImC, Fim \rangle, \quad (1.9)$$

где *RC* – базовые компоненты множества *C*; *In*-интерфейсы компонентов; *ImC* – реализации компонентов; *Fim* (…) – функции, превращающие интерфейс и множество элементов данных в сигнатуру интерфейса.

Модели предметной области

Определение 1.1. Предметная область – это совокупность понятий, концептов выделенных объектов и их функциональных характеристик.

Объекты ПрО уточняются, структурно упорядочиваются с помощью теоретических операций (объединения, селекции, пересечения и др.) и могут образовывать классы и суперклассы объектов на основании общим признакам и характеристикам [1, 2. 15-21].

Модель ПрО имеет вид:

$$M_{ПрО} = \{M_o, Mi_{uc}, M_{ox}, M_{ПС}, P, D\}, \quad (1.10)$$

где M_o – множество объектов и отношений между ними, заданных в ОМ;

Mi_{uc} – *модель* интерфейса объектов ОМ;

M_{ox} – множество общих характеристик объектов и внутренних, присущие каждому объекту, и используются при конфигурационной сборки объектов ПрО;

$M_{ПС}$ – модель программной системы, которая реализует задачи и функции ПрО;

P – множество предикатов с порядком и условиями выполнения объектов с их функциональными характеристиками модели MF и взаимосвязями

объектов ПрО, методы которых обеспечивают их программную реализацию в ПС;

D – множество данных ПрО, которые необходимы для выполнения отдельных компонентов и ПС и могут сохраняться в базах данных.

Определение 1.2. Программная система (ПС) – это совокупность различных программных артефактов (объектов, компонентов, сервисов), их реализаций и их сборка в конфигурацию системы в заданной среде. Семейство программных систем (СПС) – это совокупность ПС, которые определяются общим множеством понятий и характеристик присущих каждому отдельному члену СПС.

Модель ПС – это множество артефактов, КПИ, функций (объектов), интерфейсов и данных:

$$\mathrm{M}_{\text{пс}} = (C_L, M_f, M_s, Mi, Md), \qquad (1.11)$$

Где C_L = компонентные элементы в языках $L = L_1, L_2, ..., L_N$;

$M_f = (O_1, O_2, ..., O_r)$ – множество функциональных объектов ПрО;

$Ms = (Ms_{in}, Ms_{out}, Ms_{inout}\}$ – множество сервисов (s) по обеспечению интерфейса – входного $\mathrm{M}s_{in}$, выходного Ms_{out} и серверного Ms_{inout};

M_i – множество интерфейсов в языке IDL;

M_d – множество данных и метаданных ПС.

Определение 1.3. Вариантность – это способность готовой системы ПС к замене некоторых КПИ новыми артефактами или ПС.

Точка вариантности – это место объекта в созданной ПС. В нем осуществляется выбор варианта, который будет использоваться в процессе выполнения в системе. MF характеристика транслируется в коллекцию вариантов из заданных точек вариантности в ПС.

Определение 1.4. Точка вариантности – это место в интерфейсе модели ПС, где осуществляется выбор варианта системы. Вариантная характеристика представляет собой множество вариантов или определенное количество точек вариантности ПС.

Точки вариантности обрабатываются конфигуратором и позволяет трансформировать ПС путем замены одних КПИ другими, более функциональными или корректными.

Определение 1.5. Вариабельность – это свойство продукта (системы) к расширению, изменению, приспособлению или конфигурированию с целью использования в определенном контексте и обеспечения последующей его эволюции.

Обеспечение вариабельности ПС базируется на методах разработки и конфигурирования продукта (Product Configuration) из готовых КПИ или взятых из библиотек reuses и процессов управления конфигурацией (Configuration Management), направленных на определение и внедрение эффективных процедур разработки и эволюции систем, а также на их

адаптацию к новым условиям функционирования в современных гетерогенных средах.

Модель вариабельности ПС

$$MF_{var} = (SV; AV), \qquad (1.12)$$

где SV – подмодель вариабельности артефактов структуры ПС;

AV – подмодель вариабельности разработанного продукта ПС.

Модель MF_{var} обеспечивает уровень изменяемости артефактов и продуктов ПС и снижение затрат и уменьшение сроков разработки продукта ПС.

Подмодель $\mathrm{SV} = ((\mathrm{G_t}, \mathrm{TR_t}), \mathrm{Con}, \mathrm{Dep})$,

где $G_t = (F_t, LF_t)$ – граф артефактов типа t (требования, компоненты, тесты и др.);

TR_t – двусторонние связи артефактов типа t;

Con і *Dep* – предикаты на декартовом произведении множеств артефактов, которые задают ограничения и зависимости между функциями и показателями качества ПС.

Подмодель AV определяет структуру ПС из КПИ, которые имеют паспорта, и сохраняются в репозитории. Подмодель SV отображает функциональные и вариантные характеристики КПИ и продукта, а также аспекты отношения между ними на разных уровнях модели. Модель SV конкретизируется в специальную линию разработки КПИ и конфигурирования их в ПС.

Результатом сопоставления определений системы алгебр на каждом уровне и изоморфизма этих систем является

Теорема 1.2. Отображение объекта O в его реализацию множества компонентов C

τ: $O \rightarrow C$, является изоморфизмом объектной и компонентной модели ПС как алгебраических моделей типа ⟨2,2,2,2,2⟩

$$\langle O; CH_O, VC_O, IM_O, EQ_O, EX_O\rangle;$$
$$\langle C; CH_C, VC_C, IM_C, EQ_C, EX_C\rangle.$$

Следствие 1.1. Отображение, которое сопоставляет объекту его интерфейс (КПИ) и для которого он является входным, является изоморфизмом алгебраических моделей типа 2 (O; R), (I; R) и (I; R), (C, R). Эти модели превращают точки вариабельности / варианты сложных объектов в точки вариабельности / варианты интерфейсов и КПИ в системе.

Приведенные модели метода моделирования систем из артефактов (reuses, object, services, components) служат основой их трансформации к программному виду, который получается путем конфигурации моделируемой системы на основании **модели конфигурации** [15], имеющей вид:

$$\mathbf{M}_{\text{konf}} = \langle OM, M_{\text{ПрО}}, M_{\text{пс}}, MF_{var}, M_{\text{вз}}),$$

где $M_{\text{вз}}$ – модель взаимодействия отдельных элементов системы

На основании модели конфигурации осуществляется:

- формирование набора артефактов и ресурсов ПС для заданной ПрО;
- выделение общих и вариантных характеристик ПС для построения модели характеристик FM и модели ПрО;
- набор элементов ПС или подбор готовых ресурсов, накопление их в базе конфигурации;
- планирование многократного использования ресурсов для ПС в точках вариантности, (подобно точек use case UML или точек вариабельности в стандарте SEI IEEE- 15534), используемых для внесения или удаления некоторых функций внешними и внутренними операциями;
- реализация модели вариабельности из артефактов и ресурсов, в том числе КПИ;
- сборки ресурсов в ПС и их адаптация к новым условиям среды функционирования;
- управления вариантами ПС с заменой отдельных функций в ПС
- управление моделью взаимодействия в гетерогенной среде.

Способность к взаимодействию двух и больше программ или систем поддерживается интерфейсом, который описывается в языке IDL (Interface Definition Language). На общем уровне описание интерфейса включает данные, которые передаются другим программам для обеспечения взаимодействия (interconnection) таких разнородных программ.

Определение 1.6. *Взаимодействие* – это совместимое выполнение двух и более объектов в разных средах. Модель *взаимодействия* включает набор параметров интероперабельности программ и процессов в заданной среде.

Модель взаимодействия имеет вид:

$$Мвз = \{Мпр, Мсис, Мсред\},$$

где $Мпр = \{Com, Int, Pro\}$ – модель программного ресурса,

$Мсис = \{FPC, Int, Pro\}$ – модель системы,

$Мсред = \{Envir, Int, Pro\}$ – модель среды,

Эта модель базируется на интерфейсе и механизмах передачи данных по сети для обеспечения взаимодействия систем, миграции элементов систем или ПС из одной среды в другую и для вычисления задач на определенных данных.

В качестве примера студенты МФТИ реализовали конкретные модели взаимодействия систем *Visual Studio, Eclipse*, *CORBA,* WSphere [20, 21].

1) Visual Studio.Net↔Eclipse – это виртуальная среда для технологии разработки отдельных программ в языке C# и спецификации интерфейса для переноса готового продукта в репозиторий системы Eclipse. Эта система отображает связь с исходной средой разработки программ с помощью плагинов и конфигурационного файла с параметрами и операциями обработки данных в среде и среде выполнения Eclipse;

2) CORBA↔JAVA↔MS.Net обеспечивают разработку программ на ЯП этой среды и устанавливают связи между этими средами с целью размещения разработанных программ в репозитории Eclipse для предоставления доступа другим разработчикам к этим программам;
3) IBM WSphere↔Eclipse предоставляет средства для разработки новых программ с использованием ЯП, которые допустимы в среде или в WSphere виртуального варианта системы.

Кроме этих моделей взаимодействия программ и сред исследованы IContract средства взаимодействия WCF (Windows Communication Foundation) [18].

2. Формальный переход от объектой модели к компонетной

Одна из ключевых задач моделирования компонентной модели систем является построение КПИ, которые реализуют функции объектов и могут многократно использоваться в новой разработке ПС. Проектирование системы из компонентов – природное расширение ООП. Компоненты не являются объектами, а предоставляют им необходимые ресурсы и сервисы для отображения их функций. Сформировалась теория о компонентах, их представления в компонентном графе с интерфейсами, способах конфигурационной сборки объектов и компонентов в ПС.

Метод ОКМ обобщает понятие объектов, как элементов реального мира с их свойствами и характеристиками, которые определяются и уточняются с помощью математических формализмов отображения объектов ОМ в компоненты и интерфейсы с сохранением их интерфейсов [1, 2, 15]. В нем конфигурационная сборка программных элементов основывается формальных моделях компонента, среды и системы. Эти модели реализуются с помощью компонентной алгебры и теории интерфейсов с методами преобразования передаваемых данных с помощью элементарных функций преобразований типов данных системной сигнатуры операций.

Введение понятия компонентной алгебры

Рассмотрим множество компонентов (Component - C) $C = (c_1, c_2, \ldots c_r)$, где $c_i \in C$ – произвольный компонент, использующийся при разработке ПС. Каждый компонент характеризуется некоторой совокупностью свойств (например, расширяемость, взаимозаменяемость и др.).

Пусть $P_1, P_2, \ldots, P_k$ – предикаты свойств компонентов на множестве C, т.е.

$$P = < P_1(C), P_2(C), \ldots, P_k(C) >. \qquad (2.1)$$

Над $c_i \in C$ возможны некоторые операции, результаты выполнения которых также являются элементами множества C. Тривиальными примерами таких операций могут быть объединение нескольких объектов в один или разделение

некоторого компонента на отдельные части. Обозначим через ΣCR – алгебру, содержащую множество компонентов C и операций R над компонентами C. Каждая операция имеет собственную арность в зависимости от ее семантики. В целом существуют операции, которые задаются на подмножестве целых, а также могут в качестве результатов использоваться подмножества (например, операции объединения и разделения).

Среди множества операций рассмотрим те, которые для своих результатов сохраняют свойства компонентов. Обозначим эти операции через $R = (R_1, R_2, \ldots R_m)$. Тогда для любых $c_1, c_2, \ldots c_r$, принадлежащих C и любой R_i , $i=1,2, \ldots m$ с учетом арности операции R_i и ее результата справедливо одно из двух выражений:

$$R_i\,(c_1, c_2, \ldots c_r) \in C \quad \text{или } R_i\,(c_1, c_2, \ldots, c_r) \notin \text{C}.$$

Множество C и операции $R_1, R_2, \ldots R_m$ определяют алгебру компонентов. Примерами реальных операций над компонентами могут быть операции расширения интерфейсов, рефакторинга и др.

Множество компонентов C, рассмотренное выше, является математической абстракцией. В реальном случае, мы имеем некоторое множество компонентов $c_i \in C$.

Применив к этому множеству операции $R_1, R_2, \ldots R_m$, получим множество $C`$, которое будет называть замыканием множества $C$. Этим термином показано, что C` имеет максимально множество компонентов C и каждый $c_i \in C$.

Компонентная модель системы

Под *компонентной моделью* системы понимается такое абстрактное представление конструируемой системы, в которой элементами являются реальные компоненты, обеспечивающие реализацию функций системы и выполнения нефункциональных требований к системе. Для одной и той же системы существует множество компонентных моделей в зависимости от концепций проектирования с использованием компонентного подхода. Рассмотрим обобщенный уровень представления модели, используя функции компонентов и совокупность контрактов (интерфейсов) между ними.

Пусть $\prod A_i$ – множество интерфейсов, относящихся к определению функций компонентов. Каждому A_i можно сопоставить интерфейс I_i , который описывает интерфейс как клиент-серверное взаимодействие с соответствующими методами и структурами данных. В соответствии с этим каждому интерфейсу можно сопоставить пару In_i и Out_i, которые будем называть соответственно определяющим представлением I_i и реализующим представлением Out_i. Параметр In_i определяет условие и цель интерфейса со стороны клиента, а Out_i задает аспект реализации интерфейса со стороны сервера.

После того, как все In_i и Out_i определены, их можно группировать в различных сочетаниях для элементов C.

Рассмотрим произвольную совокупность In_i, Out_j, где $i \neq j$. В нее не входят одновременно определяющее и реализующее представления для одного и того же интерфейса.

Каждая совокупность $C_i \cup Iv = \{C_i, I_v, Out_J, Inout_{ij}\}$ входит в модель компонента или является шаблоном компонента, содержащим некоторое множество определяющих и реализующих представлений интерфейсов.

2.1. Формальные модели и операции компонентного проектирования

Для установления связей между объектно-ориентированным и компонентным моделированием рассмотрим компонентную модель системы [1, 4, 15].

Модель компонентной системы имеет вид**:**

$$M_{кс} = \{CLm\{Lm_1, \ldots, Lm_n\}, P\,\{P_i, \ldots, P_m\}, \quad CLn\,\{In_1, \ldots, In_k\}, D_i\,). \qquad (2.2)$$

где CL_m – компоненты из множества реализаций в языках L,

$P\{P_i, \ldots, P_m\}$ – множество предикатов, соответствующих процессам сборки или конфигурации КПИ в ПС на основе реализаций компонентов CLm и интерфейсов In;

CLn – множество интерфейсов компонентов;

D_i – множество данных.

Модель $M_{кс}$ состоит из множества функций реализаций (КПИ, reuses), предикатов, интерфейсов и данных.

Условие целостности $КС$ заключаются в существовании для каждого компонента C_1 из C, имеющего исходный интерфейс CIn_1^u, компонента C_2 с соответствующим входным интерфейсом CIn_2^m, и контракт $Cont_{12}^{im} = (CIn_1^u, CIn_2^m, IMap_{12}^{im})$, входящих в множество C.

Процесс построения $M_{кс}$ включает в себя создание компонентной среды, определение начальных компонентов и определенное множество интерфейсов в соответствии с функциональными требованиями к $КС$. Суть моделирования системы состоит в том, чтобы представить модель $M_{кс}$ такую, чтобы для любого элемента системы существовал компонент из C или он мог быть получен из $C`$ посредством конечного числа допустимых операций компонентной алгебры.

Компонентная модель с классами имеет вид:

$$CSyst = (CClass, G) \qquad (2.3)$$

где $CClass = \{CClass_i\}$ – множество классов, G – компонентный граф.

Интерфейсная модель в общем случае задается выражением

$$ISyst = (IFunc, IG),$$

где $IFunc = \{IFunci\}$ – множество интерфейсов, IG – интерфейсный граф, который эквивалентен графу G модели CM. Для классов $CClass$ определяются условия, когда они допускают представление их как элементов множества

интерфейсов в интерфейсном графе. Для всех классов *CM* справедлива следующая теорема.

Теорема 2.1.Существует единственное изоморфное отображение между компонентной моделью C*Syst* и интерфейсной моделью *ISyst*.

Эта теорема определяет условия существования эквивалентного отображения между объектным и компонентным представлениями системы. Как следствие, на этапах анализа и проектирования СМ могут применяться модели и методы ООП, а на последующих этапах – собственные компонентные модели.

Определение 2.1. *Программный компонент* – это самостоятельно реализованный программный объект, который обеспечивает выполнение определенной совокупности прикладных функций, доступ к которым возможен только через интерфейс и операции обращения к другим компонентам.

Компонент отображает типовое решение или функцию и при последующей композиции с другими объектами и компонентами имеет типичную интерфейсную часть для обмена данными в разных средах. То есть компонент, становится неделимым и инкапсулированным объектом, который удовлетворяет функциональным требованиям и требованиям компонентной среды.

2.1.1. Модели компонентного проектирования

К моделям относится модель компонентов, интерфейса, среды и системы. Определим их.

Модель компонента – это обобщение типовых решений, касающихся сущности объекта, их архитектуры, структуры, свойств, характеристик, которые постепенно переходят или отображаются в компонент.

Формально модель компонента имеет вид:

$$Comp = (CName, CInt, CFact, CImp, CServ), \quad (2.4)$$

где $CName$ – уникальное имя компонента;
$CInt = \{CInt_i\}$ – множество интерфейсов, связанных с компонентом;
$CFact$ – интерфейс управления экземплярами компонента;
$CImp = \{CImp_j\}$ – множество реализаций компонента;
$CServ = \{CServ_j\}$ – множество системных сервисов.

Множество $CInt=CIntI \cup CIntO$ состоит из входных $CIntI$ и исходных $CIntO$ интерфейсов. Отличие между ними заключается в том, что для входных интерфейсов компонент имеет собственные реализации, а для выходных интерфейсов реализации находятся в других компонентах. Интерфейс *CFact* определяет методы, которые необходимы для управления экземплярами компонента (поиск, создание, уничтожение и т.п.).

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

Модель интерфейса имеет вид:

$$CInt_i = (IntName_i, IntFunc_i, IntSpec_i)$$

где $IntName_i$ – имя интерфейса;
$IntFunc_i$ – функциональность (совокупность методов);
$IntSpec_i$ – спецификация интерфейса (описания типов, констант, других элементов данных, сигнатур методов и т.д.).

Необходимым требованием существования компонента является условие его целостности:

$$\forall CInt_i \in CInt_I \; \exists CImp_j \in CImp \; [Provide(CInt_i) \subseteq CImp_j],$$

где $Provide(CInt_i)$ означает функциональность, которая обеспечивает реализацию методов интерфейса $CInt_i$. Для взаимодействия двух компонентов $Comp_1$ и $Comp_2$ существует следующее необходимое условие: если $CInt_{i1} \in CInt_{01}$, то должен существовать $CInt_{k2} \in CInt_{I2}$ такой, что:

$$Sign(CInt_{i1}) = Sign(CInt_{k2}) \& Provide(CIn_{ti1}) \subseteq CImp_{j2}$$

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

Наличие знака включения в данной формуле означает, что избранная реализация компонента может обеспечить поддержку не только необходимого интерфейса, но и других возможностей. Технологии и ЯП (CORBA, COM, JAVA, C++ и др.) содержат необходимые средства. Интерфейс может иметь несколько реализаций, которые различаются особенностями функционирования (например, операционной средой, средствами хранения данных и т.д.).

Функциональные элементы компонентной модели могут быть равными, эквивалентными или подобными.

Определение 2.2. Два компонента $Comp_1$ и $Comp_2$ являются тождественными (равными), если тождественны их соответствующие составные. Как следствие, замена $Comp_1$ на $Comp_2$ *не* влияет на компонентную систему , к которой принадлежит $Comp_1$.

Определение 2.3. Два компонента $Comp_1$ и $Comp_2$ являются эквивалентными, если эквивалентны их соответствующие составные, кроме имен компонентов. Замена $Comp_1$ на $Comp_2$ *не* меняет функциональности компонентной системы при условии установления соответствия между именами в самой системе.

Определение 2.4. Два компонента $Comp_1$ и $Comp_2$ подобны, если эти компоненты имеют тождественное множество интерфейсов. Замена $Comp_1$ на $Comp_2$ сохраняет начальную функциональность компонентной системы, к которой можно добавить новые функциональные свойства.

Главной средой использования компонентов является ***компонентная среда***, модель которой имеет вид:

$$CE = (NameSpace, IntRep, ImpRep, CServ, CServImp), \quad (2.5)$$

где $NameSpace = \{CName_m\}$ – множество имен компонентов среды;
$IntRep = \{IntRep_i\}$ – репозиторий интерфейсов компонентов среды;
$ImpRep = \{ImpRep_j\}$ – репозиторий реализаций.

CServ = {CServ$_r$} – интерфейс множества системных сервисов;

CServImp = {CServImp$_r$} – множество реализаций для системных сервисов.

Модель компонентной среды рассматривается как множество серверов приложений, где разворачиваются компоненты–контейнеры, экземпляры которых обеспечивают реализацию функциональности компонента. Взаимосвязь контейнера с сервером обеспечивается через стандартизированные интерфейсы *(CFact)*. Связь между компонентами, которые развернуты в разных серверах, обеспечивается реализация интерфейса *CServ*.

Определение 2.5. Каркасом компонентной среды называется среда, для которой совокупность имен компонентов, интерфейсов и реализаций – суть пустые множества, то есть

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

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

Определение 2.6. Каркас FW_1 совместим с каркасом FW_2, если существует отображение $SMap: CServ_1 \rightarrow CServ_2$ такое, что $SMap(CServ_1) \subseteq CServ_2$.

Совместимость каркасов и компонентных сред означает, что среду $CSer_1$ должен входить сервис именования, а в $CServ_2$ – аналогичный сервис и между этими сервисами должна существовать такая связь, при которой сервис среды – это именование компонентов среды и наоборот. Аналогичная ситуация должна существовать и для других системных сервисов, так как любая компонентная среда использует те же сервисы, а их связи определяют совместимость. В частности, взаимодействие компонентов, расположенных в разных серверах, поддерживают сервисы этих серверов. Если совместимость между серверными сервисами существует, то такие компоненты могут входить в состав *CM*.

2.1.2. Операции над компонентами в компонентной среде

Далее рассматривается набор операторов, задаваемых математическими операциями $\{\cup, \cap, \oplus, \Diamond, -\}$ для задания и обработки компонентов в компонентной среде.

Операция добавления компонента к компонентой среде обозначается $\oplus$. Добавление компонента к среде выполняется согласно правил $C \oplus CE_1 = CE_2$,

$$CE_2.NameSpace = \{C.CName\} \cup CE_1.NameSpace,$$
$$CE_2.InRep = \{C.(CIn_i, CName)\} \cup CE_1.InRep,$$
$$CE_2.ImpRep = \{C.(C_j, CName)\} \cup CE_1.ImpRep. \quad (2.6)$$

Аксиома 2.1. Операция добавления компонента к компонентной среде коммутативна:

$$C \oplus CE = CE \oplus C.$$

Аксиома 2.2. Операция добавления компонента к компонентной среде ассоциативна:

$$C_1 \oplus (CE \oplus C_2) = (C_1 \oplus CE) \oplus C_2.$$

***Утверждение* 2.1.** Каждая непустая компонентная среда CE может быть представлена в виде: $CE = C_1 \oplus C_2 \oplus , ..., \oplus C_n \oplus FW$.

Операция удаления компонента из компонентной среды обозначается знаком $\Diamond$ и подчиняется следующим правилам:

$$CE_1 \Diamond C = CE_2, \quad (2.7)$$

$$\exists CName_m: CName_m \in CE_1.NameSpace \cap (CName_m = C.CName) \Rightarrow$$
$$CE_2.NameSpace = CE_1.NameSpace \setminus \{C.CName\} \cap CE_2.IntRep = CE_1.IntRep \setminus$$
$$\{(\forall i: IntRep^i.CName = C.CName)\ IntRep^i\} \cap CE_2.ImpRep = CE_1.ImRep \setminus$$
$$\{(\forall j\ \&ImtRep^j.CName = C.CName)\ ImRep^j\}.$$

Теорема 2.2. Для любого компонента *C* и компонентной среды *CE* выполняется равенство $(C_2 \oplus CE) \Diamond C = CE$.

Доказательство. Представим операцию добавления компонента к среде так:

$$C \oplus CE = (\{C.CName\} \cup CE.NameSpace\},\ \{C.(CIn_i, CName)\} \cup CE.InRep,\ \{C.(CIm_j, CName)\} \cup CE.ImRep,\ CSe,\ CSeIm) = CE.$$

Тогда операция удаления компонента из среды имеет вид:

$$CE' \setminus C = (CE'.NameSpace \setminus \{C.CName\},$$
$$\{CE'.InRep \setminus C.(CIn_i, CName)\},$$
$$CE'.ImpRep \setminus \{C.(CIm_j, CName)\},\ CSe,\ CSeIm) = CE.$$

Операция замены компонента задается знаком "–" и имеет вид:

$$CE.NameSpace(C_1) - C_2 = (CE \Diamond C_1) \oplus C_2. \quad (2.8)$$

Операция объединения (обозначается $\cup$) компонентных сред и выполняется согласно следующих правил:

$$CE_1 \cup CE_2 = CE_3, \quad (2.9)$$
$$CE_3.NameSpace = CE_1.NameSpace \cup CE_2.NameSpace,$$
$$CE_3.InRep = CE_1.InRep \cup CE_2.InRep,$$
$$CE_3.ImRep = CE_1,\ ImRep \cup CE_2.ImRep.$$

Теорема 2.3. Операция объединения компонентных сред ассоциативна:

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

Доказательство

$$\forall C \in (CE_1 \cup CE_2) \cup CE_3 \Rightarrow C \in CE_1 \cap C \in CE_2 \cap C \in CE_3;$$
$$\forall C \in CE_1 \cup (CE_2 \cup CE_3) \Rightarrow C \in CE_1 \cap C \in CE_2 \cap C \in CE_3.$$

Аксиома 2.3. Операция объединения компонентных сред коммутативна:

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

Утверждение 2.2. Для любых компонентных сред выполняется:

$CE \cup FW = FW \cup CE = CE$.

Утверждение 2.3. Для любой компонентной среды CE∪FW = FW ∪ CE = CE.

Утверждение 2.4. Для двух произвольных компонентных сред CE_1, CE_2 и компонента C всегда выполняется:

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

Утверждение 2.5. Для любого компонента C и компонентной среды всегда выполняется соотношения: $C \in CE$.

Процесс построения СМ включает в себя создание компонентной среды, определение начальных компонентов и формирование множества контрактов в соответствии с функциональными требованиями к компонентной системе.

2.1.3. Операция добавления реализации и интерфейса компонента

Особенность операции добавления реализации компонентов (*addImp)* заключается в том, что множество интерфейсов компонента расширяться за счет добавления новых входных интерфейсов, связанных с реализованной дополнительной функциональности нового компонента реализации.

Пусть имеем дополнительное множество исходных интерфейсов:

$NewIntOs = \{NewIntOsq\}$.

В частном случае $NewIntOs = \varnothing$, если не требуется добавлять дополнительную реализованную функциональность.

Рассмотрим две разновидности этой операции: *добавление AddOImp* существующего интерфейса и нового интерфейса:

NewComp = AddImp(OldComp, NewCImps,NewCIntOs) и представление семантики

$NewInt = OldInt \cup NewIntOs$

$NewCImp = OldCImp \cup \{NewImps\}(\exists\ OldIntt \in OldCIntI)\ Provide(OldInt) \subseteq NewImps,$

где *NewCImps* – реализация, которая добавляется;

OldCInt – множество существующих интерфейсов.

Условие целостности компонента выполняется автоматически, потому что множество входных интерфейсов остается прежним. Из целостности старого компонента вытекает целостность нового компонента.

Операция *AddImp* является ассоциационной и коммутативной над множествами компонентов. Доказательство этих фактов вытекает из анализа множеств интерфейсов и множества реализаций, которые входят в состав соответствующих множеств компонентов.

Вторую разновидность операции добавления реализации представим как *AddNImp* в форме

NewComp = AddNImp(*OldComp,NewCImps,NewCIntOs*) и семантикой

$NewCInt = OldCInt \cup NewCIntOs$

$NewCImp = OldCImp \cup \{ NewCImps\}$.

где *NewCImps* – реализация, которая добавляется к множеству реализаций. Как и предыдущая, данная операция обеспечивает целостность компонента и обладает свойством ассоциативности и коммутативности.

Операция замены существующей реализации новой *ReplImp* имеет вид:

NewComp=ReplImp(*OldComp, NewCImps, NewCIntO*s *OldCImpr, OldCIntOr*) с семантикой.

Если справедливо

$(\forall OldCIntt \in OldCInt) \& (Provide(OldCIntt) \subseteq OldCImpr) \Rightarrow$

$(Provide(OldCIntt) \subseteq NewCImps) \vee ((\exists OldCImpj \in$

$(OldCImp \setminus \{OldCImpr\})) \ \&\ Provide(OldCIntt) \subseteq OldCImpj)$, то

$NewCInt = OldCInt \cup NewCIntOs \setminus OldCIntOr$;

$NewImp = OldImpl \cup \{NewCImps\} \setminus \{OldCImpr\}$,

где *NewCImps* – реализация, которая добавляется;

NewCIntOs – множество дополнительных исходных интерфейсов для реализации, которая добавляется;

OldCImpr – реализация, которая замещается;

OldCIntOr – множество исходных интерфейсов, связанных с реализацией, которая замещается.

Лемма 2.1. Операция замещения реализации компонента новой семантикой сохраняет условие целостности компонента.

Для любого входного интерфейса, создаваемого компонентом, кроме интерфейсов, соответствующей реализации, которая замещается *OldCImpr* – условие целостности выполняется. Для интерфейсов, отвечающих реализации *OldCImpr,* справедливо, что $Provide(OldCIntt) \subseteq NewCImps$ существует для реализации базового компонента. Объединяя эти два случая, получаем, что для любого входного интерфейса создаваемого компонента выполняется условие целостности.

Операция *расширения* существующей реализации семантики эквивалентна операции замещения, где исходная реализация замещается, а расширенная – добавляется. Потому нет необходимости вводить отдельную операцию.

Операция добавления интерфейса. Исходный интерфейс может добавляться путем замены существующей реализации или новой реализации. Операция *добавления* исходного интерфейса – это составная операция добавления реализации.

Операция добавления нового входного интерфейса *AddInt* имеет вид:

NewComp = AddInt(*OldComp, NewCIntIq*) с семантикой согласно следующему правилу.

Если справедливо, что

$$(\exists\ OldCImps \in OldCImp)\ \&\ (Provide(NewCIntIq) \subseteq OldCImps),$$

то NewCInt = OldCInt $\cup${NewCIntIq}, NewCImp = OldCImp, где NewCIntIq – новый интерфейс.

Лемма 2.2. Операция добавления интерфейса с заданной семантикой сохраняет условие целостности компонента.

Пусть выполняется условие целостности для базового компонента, т.е. для каждого из входных интерфейсов существует соответствующая реализация. Эта предпосылка требует наличия реализации и для нового интерфейса. Потому расширенное множество входных интерфейсов целостности компонента истинно для созданного компонента с сохранением целостности.

Для операции расширения существующего интерфейса в отличие от операции расширения существующей реализации, не требующей своей сохранности для структуры компонента, все существующие входные интерфейсы должны сохраняться. Операция *CFact* носит комплексный характер, потому что дополнительные методы, которые входят в состав интерфейса, требуют реализации со стороны контейнера.

Модель сервиса детализирует *CServ* сервисов, которые необходимы для поддержки функционирования компонентов и компонентных сред в рамках парадигмы компонентного программирования. Она обеспечивает формальное определение статических и динамических свойств компонентных сред.

2.2. Компонентная алгебра

Для трансформации моделей компонентного проектирования к выходному коду разработана компонентная алгебра – внешняя, внутренняя и эволюционная алгебры [15]**:**

$$\sum = \{\varphi_1, \varphi_2, \varphi_3\}, \qquad (2.10)$$

где φ_1= {*CSet, CESet,* Ω_1} – внешняя алгебра, φ_2 = {*CSet, CESet,* Ω_2} – внутренняя алгебра, φ_3 = {*Set, CESet,* Ω_3} – алгебра эволюции компонентов.

Внешняя компонентная алгебра

Эта алгебра двуосновная и имеет следующий вид:

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

где *CSet* – множество компонентов С; *CESet* – множество компонентных сред; Ω_1– множество операций над этими элементами.

Операции над компонентами такие:

- инсталляция (развертывание компонента) $CSet_2 = Cset \oplus CESet_1$;
- объединение компонентных сред $CESet_3 = CESet_1 \cup CESet_2$;
- удаление компонента из компонентной среды $CESet_2 = CESet_1 \setminus CSet$.

Внутренняя компонентная алгебра –

$$\varphi_2 = \{CSet, CESet, \Omega_2, \Omega_3\}, \qquad (2.11)$$

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

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

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

Ω_2 = {*addImp, addInt, replInt, replImp}* включает операции:

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

Ω_3 = {O_{refac}, O_{Reing} , O_{Rever}} – алгебра эволюции (φ_3),

где O_{refac} – рефакторинг для построение компонентной ПС и возможностью внесения изменений;,

O_{Reing} – реинженеринг для обеспечения замены, переименование компонентов и/или их интерфейсов, а также добавление новых компонентов в ПС и в компонентную среду;

O_{Rever} – реверсная инженерия обеспечивает восстановление структуры компонента по его выходному коду.

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

Модель рефакторинга компонентов:

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

где O_{refac} = {AddOImp, AddNImp, ReplImp, AddInt} – операции рефакторинга,

пара (*CSet,* O_{refac}) – элемент компонентной алгебры эволюции.

Операция рефакторинга носит комплексный характер, так как методы, входящие в этот интерфейс, требуют реализации компонента в составе контейнера. Поэтому реализация этой операции связана с существованием нового типа контейнера. Согласно классификации методов рефакторинга, операция расширения интерфейса управляет экземплярами компонентов и относится к расширенной классификации. Множество операций рефакторинга *AddOImp, AddNImp, ReplImp, AddInt* включают в себя операции добавления и замещения реализаций компонентов и интерфейсов.

Модель реинженерии компонентов:

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

где O_{reing}= {rewrite, restruc, adop, supp, conver } – операции реинженерии,

пара (*CSet,* O_{Reing}) – элемент компонентной алгебры эволюции.

Алгебра реинжиниринга $\Sigma^{Reing} = (CSet, Reing)$ используется при условии нарушения целостности компонентов или изменения функциональности. Эта алгебра может быть построена лишь при условии нарушения целостности представления компонентов. Кроме операций рефакторинга (*Refac*), во множество *Reeng* входят операции, которые удаляют из компонента существующий интерфейс или изменяют его сигнатуру. Это усиливает условие целостности, так как другие компоненты, которые обращаются к нему, не могут иметь доступа к необходимой функциональности. Исходя из таких условий, вместо алгебры реинжиниринга в состав формальных методов компонентного программирования целесообразно включить модель реинжиниринга.

Модель реверсной инженерии компонентов:

$$M_{Rever} = \{O_{Reve}, , \{CSet = \{NewComp^n\}\},$$

где O_{Rever}= {restruc, design}– операции реверсной инженерии;

пара $(CSet, O_{Reing})$ – элемент компонентной алгебры эволюции.

Алгебра реверсной инженерии $\Sigma^{Rever} = (CSet, Revera$ используется при условии восстановления структуры и спецификации ПС. В ней могут использоваться операции реинженерии $Reeng \subset Revers$. Особенность *Revers* состоит в том, что ее операции не определены полностью, а лишь частично. Реверсная инженерия обозначает полную перестройку ПС из компонентов в некоторой среде, включая изменение отдельных показателей качества компонентов, которые могут изменяться в зависимости от смысла функции и применяемых операций изменения компонента. Семантика операций *Reeng* может определять трансформацию не только целевого компонента, но и других связанных компонентов. Например, при изменении сигнатуры входного интерфейса необходимо одновременно изменить другие компоненты, которые содержат обращения к методам этого интерфейса.

Таким образом, операции эволюционной алгебры обеспечивают изменения конкретного компонента и относятся к классу операций в других методах программирования, которым соответствует операциям типа *эволюции*. Базовая суть этих операций – изменение имен, интерфейсов, реализаций и связей между элементами, а также операции преобразования структуры и функций компонентов.

2.3. Связь компонентной и объектной моделей

Результатом сопоставления определений системы алгебр и изоморфизма этих систем является

Теорема 2.4. Отображение О→С ,создает сложный объект из КПИ и его реализации, изоморфизм объектной и компонентной подмодели СПП, как алгебраических моделей типа <2,2,2,2,2>

Отображение, которое сопоставляет объекту его интерфейс, а этому интерфейсу – КПИ, для которого он является входным, – изоморфизм алгебраических моделей типа 2 (*O*; *RO*), (*I*; *RI*) и (*I, RI*), (*C, RC*), которые

превращают точки вариабельности / варианты сложных объектов в точки вариабельности / варианты интерфейсов и КПИ.

Связь компонентной модели с объектной ОМ проследим с помощью процесса функционирования компонентов *Create* и интерфейса *Cfac***,** порождающего экземпляры компонентов:

$$Cfact.Create: Comp \rightarrow \{Cins_k^{ij}\},\ Cins_k^{ij} = (Iins_k^{ij}, IntFunc^i, ImpFunc^j),$$

где $Cins_k^{ij}$ – экземпляр *k* компонента, который предоставляет свою функциональность с помощью интерфейса $IntFunc^i$ и обеспечивает реализацию этого интерфейса с помощью $ImpFunc^j$; $Iins_k^{ij}$ – уникальный идентификатор экземпляра компонента.

Пусть существует некоторая компонентная система, представленная диаграммой классов объектов*:*

$$OCsyst = (OClass, G), \qquad (2.12)$$

где $OClass = \{Oclass_i\}$ – множество классов; *G* – объектный граф, отражающий связи и отношения между классами и экземплярами.

Каждый класс представлен в виде:

$$OClass_i = (ClassName_i, Method_i, Field_i\},$$

где $ClassName_i$ – имя класса; $Method_i = \{Method_j^i\}$ – множество методов; $Field_i = \{Field_n^i\}$ – множество переменных, определяющих состояние экземпляров класса.

Переход от объектной к компонентной структуре связан с преобразованием объектов и интерфейсов ОМ к компонентам и интерфейсам (рис. 5). В нем показаны структура компонентов $Comp_1$ и $Comp_2$, соответствующие интерфейсы. Int_1, Int_{O1} и Int_{I2}, $IntI_{O2}$ в классе компонентов $Oclass_1$, $Oclass_2$ и $Oclass_3$.

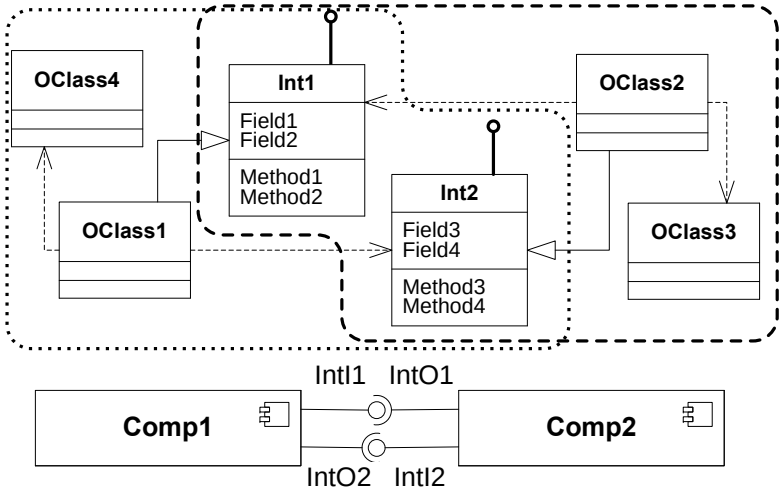


Рис. 5 – Структура связи компонентов и интерфейсов в классе

Каждый *Oclass* представляется как модель класса объектов

$$Oclass_i = \{ClassName_i, Method_i, Field_i\},$$

где $Oclass = \{Oclass_i\}$ – множество классов; $ClassName_i$ – имя класса; $Method_i = \{Method_j^i\}$ – множество методов; $Field_i = \{Field_n^i\}$ – множество переменных, определяющих состояние экземпляров класса.

Пусть $Pfield_i \subset Field_i$ – множество внешних переменных (public), доступные извне. Каждому $Pfield_n^i \in Pfield_i$ ставятся в соответствие методы *get* <$Pfield_n^i$> и *set* <$Pfield_n^i$> для присвоения и выборки значений переменной. Другими словами, эти переменные становятся атрибутами в современных компонентных моделях.

В других классах вместо непосредственного обращения к таким переменным будут использоваться указанные методы.

Введем новое множество методов:

$$Imethod_i = Method_i \cup \{\text{get} <Pfield_n^i>\} \cup \{\text{set} <Pfield_n^i>\},$$

которым сопоставим интерфейс $Ifunc^i$, состоящий из прототипов методов, которые входят в $Imethod^i$.

Вместе с *Osyst* рассмотрим систему: *Isyst* = (*Ifunc, IG*),

где *Ifunc* = {$Ifunc_i$} – множество интерфейсов; *IG* – интерфейсный граф, идентичный графу *G*.

Класс объектов $Oclass_i$ порождает свои экземпляры (объекты)

$$Obj_k^i = \{ObjName_k^i, Method_i, Field_i\},$$

которым в системе *Isyst* будут соответствовать интерфейсные элементы $Iobj_k^i = \{Iname_k^i, Ifunc^i\}$.

Для каждого такого элемента не определена реализация соответствующего интерфейса. Сопоставив некоторому интерфейсу реализацию $ImpFunc^j$ (которая обеспечивает выполнения методов интерфейса), сформируем элемент $Iobj_k^{ij} = \{Iname_k^i, Ifunc^i, ImpFunc^j\}$, который, по своей сути, эквивалентен экземпляру компонента $Cins_k^{ij} = (Iins_k^{ij}, IntFunc^i, ImpFunc^j)$.

Основные расхождения определяются следующими факторами. Во-первых, в процессе развертывания требуется выбор реализации. Во-вторых, экземпляр объектного класса порождается его описанием и не может содержать элементов больше, чем существует в самом классе или его суперклассах. Противоположно этому, реализация компонента может поддерживать несколько не связанных между собою интерфейсов. Эти две системы (объектная и интерфейсная) эквивалентны.

Таким образом, при построении компонентной системы используются соответствующие инструментальные средства. Согласно логико-математическому проектированию создается ОМ и соответствующая интерфейсная система без конкретизации реализации этих интерфейсов. Результат такого проектирования – совокупность интерфейсов, для которой рассматривается задача покрытия интерфейсов соответствующими компонентными реализациями. При этом нет необходимости учитывать

реализацию функциональности ПС в виде конфигурации компонентов, которая выполняется путем сборки компонентов в стандартной форме и развертки ПС.

Типы отношений между компонентами в ОКМ задаются в виде:

$$Comp_n=(Cname_n, Ci_n, Cfact_n, Cimp_n, Cserv). \quad (2.13)$$

К ним относятся следующие отношения:

наследование двух компонентов определяется установлением отношения наследования для входных интерфейсов (как наследования в ООП);

экземпляризация создается соответственно определенному входному интерфейсу $CIntI^i$. Выражение $CIns_k^{ij} = (IIns_k^{ij}, IntFunc^i, ImpFunc^j)$ описывает экземпляр компонента *Comp*. Здесь $IIns_k^{ij}$ – уникальный идентификатор экземпляра, $IntFunc^i$ – функциональность интерфейса $CIntI^i \in CInt$, $ImpFunc^j$ – программный элемент, который обеспечивает реализацию $CImp^i \in CImp$**;**

интерфейс между компонентами $Comp_1$ и $Comp_2$ в виде выражения $Cont_{12}^{im} = (CInt_1^u, CInt_2^m, IMap_{12}^{im})$, где $CInt_1^u \in CInt_1$ – исходный интерфейс первого компонента, $CInt_2^m \in CInt_2$ – входной интерфейс второго компонента; $IMap_{12}^{im}$ – отображение соответствия между методами, входящими в состав обоих интерфейсов с учетом сигнатур и типов данных, которые передаются. Отношения контракта существует, если компонент $Comp_2$ имеет реализацию интерфейса $CInt_2^m$, выполняющего функциональность $IntFunc_1$ и интерфейс $CInt_1^n$;

связывание компонентов $Comp_1$ и $Comp_2$ с помощью отношения контракта $Cont_{12}^{im}$, в котором между экземплярами $CIns_{1k}^{ij} = (IIns_{1k}^{ij}, IntFunc^i, ImpFuncj)$ и $CIns_{2p}^{mq}=(IIns_{2p}^{mq}, IntFunc^m, ImpFunc^q)$ существует отношение связывания через контракт $Cont_{12}^{im}$, который описывается выражением *Bind* ($IIns_{1k}^{ij}$, $IIns_{2p}^{mq}$, $Cont_{12}^{im}$).

Между объектным и компонентным представлениями систем существует неоднозначность, которая порождается тем, что определенный компонент может иметь реализации для нескольких интерфейсов *Isyst*. В случае, если каждый из интерфейсов реализуется отдельным компонентом, существует единое эквивалентное отображение между объектными и компонентными представлениями.

Пример объектного и компонентного описания сложной системы из объектов и компонентов показан на рис. 6.

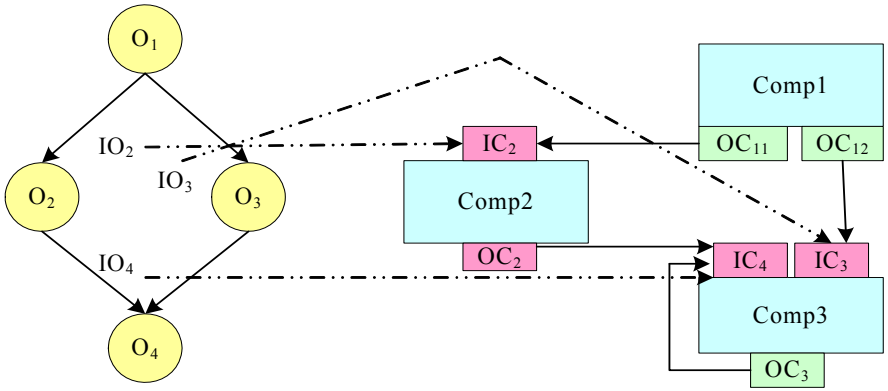
В нем используются:

$Comp1(OC_{11}, OC_{12})$, $Comp2$ (IC_2, OC_2), $Comp3$ (IC_3, IC_4, OC_3) – компоненты предметной области с входными и выходными параметрами;

O_1, O_2, O_3, O_4 – объекты класса ОМ,

IO_2, IO_3, IO_4 – входные объектные интерфейсы;

IC_2, IC_3, IC_4 – входные параметры компонентов (штрихпунктирные линии).



Рис,6 – Структура программы в ОКМ

В интерфейсном представлении *ISyst* существуют компонентные интерфейсы IC_2, IC_3, IC_4 (штрихпунктирные линии). При этом OC_{11}, OC_{12}, OC_2, OC_3 – исходные интерфейсы в компонентной модели. Между объектными и компонентными интерфейсами установлено однозначное отображение, а для объектной и компонентной модели такого отображения не существует, так как компонент $Comp_3$ имеет два входных интерфейса, т. е. функциональность классов O_3 и O_4 реализована в одном компоненте.

В ОКМ объектный анализ – это первая фаза объектно-компонентного метода проектирования ПС, при котором выявляются объекты и строится ОМ, которая адекватно отображает ее структуру, объекты и отношения между ними и т. п. Главная задача второй фазы – проектирование конкретных компонентов и ПС по результатам моделирования ПрО. В ОКМ процессы определения объектов начинаются с отдельных сущностей ПрО и заканчиваются заданием компонентов с учетом их поведения в компонентной среде. Объектная модель отображается в компонентную модель путем формирования функциональных интерфейсов и распределения их между конкретными компонентами на основе компонентной алгебры.

3. Технология конфигурационной сборки объектов и компонентов

Объекты и компоненты предложенной моделей ПС, СМ и вариабельности MF задаются специальными операциями, которые трансформируются в программную реализацию в стандартной форме WSDL и сохраняются в репозитории ИТК. Каждая из моделей ОМ, ПС и MF используются при конфигурировании артефактов и компонентов в выходную систему. Всем им соответствуют описания алгоритмов функций и данных, которые могут быть фундаментального типа (FDT), общего GDT (ISO/IEC 11404) и нестандартных типов данных (Big Data). Данные используются для обмена разных компонентов между собой через интерфейс, описанный в IDL [14, 18]. По

современным меркам каждый программный ресурс или КПИ многоразового применения должен иметь сертификат (паспорт, описание функций и типов данных (ТД) передаваемых другим компонентам через интерфейс). **Паспорт** – это информационная часть, которая в языке WSDL содержит (рис.7):

- название функции;
- ID – идентификатор ресурса;
- содержание (функции);
- параметры вызова других программ;
- инструменты поддержки выполнения программы;
- необязательные атрибуты (дата, состояние, версия, право доступа, автор, дата создания, срок пригодности, правила приобретения и т. п.).

Стандартная форма: все артефактов системы, сохраняются в репозитории и извлекаются оттуда, когда их надо использовать. По имени ресурса осуществляется поиск и анализ применимости ресурса во вновь создаваемой системе методом конфигурационной сборки.

Имеется некоторая проработка механизмов сборки в рамках интегрированной среды веб-сайта ИТК [10]. В нее включены готовые системные компоненты – Eclipce, Protege, VS.Net, CORBA, JAVA, системные компоненты (рис.7) для разработки компонентов и приложений из КПИ. При этом могут использоваться и готовые артефакты Интернет.

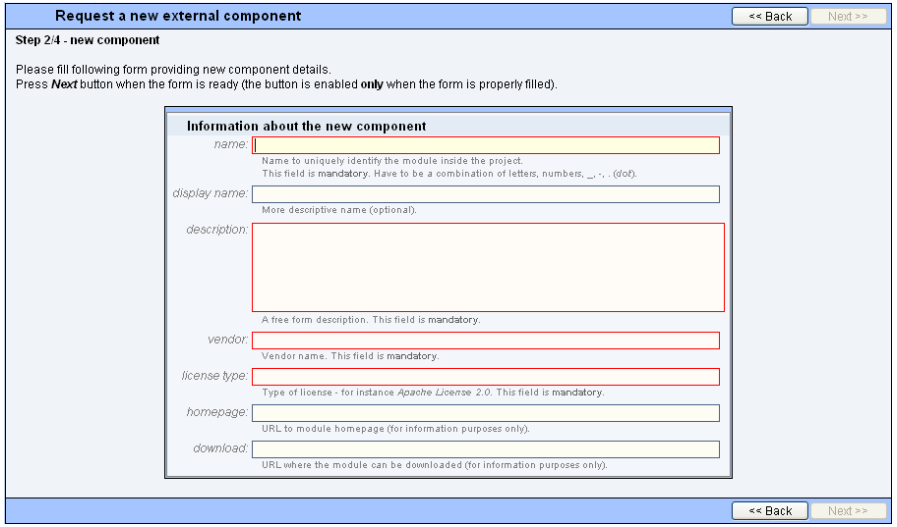


Рис. 7. Формат стандартного описания паспортных спецификаций (GRID формат)

Вариант приведенных моделей реализован с помощью следующих системных компонентов [3, 10]:

- ведение репозитория – составление сертификата в WSDL и занесение компонента в хранилище;
- поиск (выбор) компонента пользователем и анализ смысла функции для возможного применения в своих целях;
- сборка отдельных КПИ и их конфигурирование в вариант некоторой системы средствами конфигуратора;
- формальное описания некоторого артефакта в DSL и его реализация в системах DSL Tools VS.Net / Eclipse-DSL в код XML и оформление его паспорта в стандартном виде для занесения в репозиторий;
- онтологическое описание доменов с заданием модели характеристик сущностей и создания архитектуры системы из готовых объектов и компонентов, а также экспериментальная реализация онтологии доменов ЖЦ стандартов ISO/IEC 12207 и вычислительной геометрии средствами Protégé;
- трансформация описаний элементов моделей ОМ и СМ к виду DSL, ЯП и создание набора операций для их конфигурации в систему;
- тестирование разнородных ресурсов в ЯП, их объединение для комплексного тестирования системы;
- оценка стоимости (метод COCOMO II) и качества (ISO/IEC 9126) продукта с использованием результатов тестирования и испытания системы;
- обработка интерфейсных данных передаваемых между компонентами и трансформация несовместимых типов данных, которые могут иметь разные форматы данных, требующие их преобразования с помощью примитивных интерфейсных функций стандарта ISO/IEC 11404;
- новые механизмы взаимодействия программ и сред с помощью реализованных студентами трех моделей CORBA ↔ Eclipse-JAVA, VS.Net C# ↔ Eclipse, Basic ↔ C++ [22, 23].;
- обучение программированию в C# VS.Net и JAVA, а также всем аспектам курса "Программная инженерия", представленного в электронном виде студентами на веб-сайти КНУ http://programsfactory.univ.kiev.ua.

Данная технологий предназначена для разработки программных систем из готовых артефактов. Сайт ИТК (http://sestudy.edu-ua.net) пользуются по оценкам Google статистики более 100 000 респондентов, а также преподаватели ряда университетов (Украины, России, Европы, США, Канады). Основное его назначение – подготовки бакалавров и магистров по специальности программная инженерия.

4 Заключение

Рассмотрен формальный аппарат объектно-компонетного моделирования предметной области на четырех уровнях проектирования (обобщенного,

структурного, характеристи-ческого, поведенческого). Результатом обобщенного проектирования является граф объектной модели, содержащей объекты и интерфейсы. На характеристическом уровне формируется модель вариабельности, взаимодействия и конфигурирования ПС и СПС из готовых артефактов. Определены формальные модели компонентного проектирования (компонент, интерфейс, среда, система), операции объединения, замены, удаления и трансформации объектной модели к компонентной модели с учетом модели вариабельности и интерфейса, а также проведения конфигурационной сборки объектов и компонентов с помощью моделей ОМ, СМ, FM в выходной код системы. Приведены средства поддержки ОКМ на новом сайте http://www.ispras.ru/lavrischeva/sestudy.

Список литературы

[1]. Лаврищева Е.М., Грищенко В.Н. Методы и средства объектно-компонентного программирования // Кибернетика и системный анализ, 2003.– №1.– С.39–55.

[2]. Лаврищева Е.М., Грищенко В.Н. Связь разноязыковых модулей в ОС ЕС //М.: Финансы и статистика, 1982.–127 с.

[3]. Андон Ф.И., Лаврищева Е.М. Методы инженерии компьютерных приложений. Объектный подход //К.: Наук. Думка, 1997.–229 С.

[4]. Грищенко В.Н. Метод объектно-компонентного проектирования программных систем // Проблемы программирования. – 2007. – №2. – С.113–125.

[5]. Грищенко В.Н. Теоретические и прикладные аспекты компонентного программирования// Автореферат докт. диссертации, Киев, ИК НАНУ.–2007.–34с.

[6]. Лаврищева Е.М. Методы программирования. Теория, инженерия, практика // Наук. Думка, 2006.– 451 С. (www.twirpx.com)

[7]. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. –Наук.думка, 1991.-216 С.

[8]. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. Основы индустрии программных продуктов.–К.: Наук. Думка, 2009.–371 С.

[9]. Лаврищева Е.М. Колесник А.Д., Коваль Г.И., Слабоспитская О.А. Теоретичные аспекты управления вариабельностью в семействах программных систем. – Вестник КНУ, серия физ. –мат. наук. –2011, №1, с.151 –158 (укр.).

[10]. Лаврищева Е.М., Колесник А.Л., Стеняшин А.Ю. Объектне-компонентное проектироввание программных систем. Теоретические и прикладные вопросы – Весник КНУ, серия физ.-мат.наук. – 2013. – №4. – С. 150 – 164.

[11]. Слабоспитская О.Л. Модели и методы экспертного оценивания в ЖЦ ПС//Автореф. канд.дис. – Киев, ИК НАНУ.–2008.–22 С.

[12]. Ekaterina Lavrischeva, Andrey Stenyashin, Andriy Kolesnyk, Object-Component Development of Application and Systems. Theory and Practice, Journal of Software Engineering and Applications, 2014, 7, Published Online August 2014 in SciRes http://www.scirp.org/journal/jsea

[13]. Лавріщева К.М., Слабоспицька О.О., Підхід до побудови об'єктно-компонентної моделі сімейства програмних продуктів// Проблемы програмування, 2013.–№3.– с14–26.

[14]. Лаврищева Е.М., Зинькович В.М., Колесник А.Л. и др. Инструментально-технологический комплекс разработки и обучения приемам изготовления

программных систем.– Гос. служба интеллектуальной собственности Украины.– Свидетельство о регистрации авторского права. – № 45292, от 27.08.2012.
[15]. Лаврищева Е.М. Software Engineering компьютерных систем. Парадигмы, Технологии, CASE-средства программирования. –Наук. Думка, 2014. –284с.
[16]. Kolesnyk A., Slabospitskaya O. Tested Approach for Variability Management Enhancing in Software Product Line. – In: ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge, Proc. 8 –th Int. Conf. ICTERI 2012, Kherson, Ukraine, June 6 –10, 2012, CEUR –WS.org/Vol –848, ISSN 1613 –0073, urn:nbn:de:0074 –848-8. – P. 125 –133.
[17]. Колесник А.Л. Модели и методы разработки семейства вариантных программных систем. – Автореф. дис, КНУ, 2013. –22с. (укр).
[18]. Lavrischeva K., Aronov A., Dzubenko A. Programs Factory – A conception of Knowledge Reprezentation of Scientifical Artifacts From Standpoint of Software Engineering.– Computer and Information Science, Canadion Center of Science and Education.–2013.–ISSN 1913 –8989 (Print).–P.21–28.
[19]. Лаврищева. Е.М. Компонентная теория и коллекция технологий разработки индустриальных приложений из готовых ресурсов, Труды Четвертой научно- конференции «Актуальные проблемы системной и программной инженерии», АПСПИ –2015, 20 –21мая 2015, с 101 –119.
[20]. Островський А. И. Подход к обеспечению взаимодействия программных сред JAVA и Ms.Net. – Проблемы программирования, 2011.–№ 2.–с. 37–44.
[21]. Радецький І. О. Один из подходов обеспечения взаимодействия сред MS.Net і Eclipse // Проблемы программирования, № 2, 2011.– с. 45–52.

The theory of object-component modeling for software systems

E.M. Lavrischeva <lavr@ispras.ru>
ISP RAS, 25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation
Moscow Institute of Physics and Technology, 9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia

Abstract. The paper presents the formal mechanism for object-component modeling of a subject area, including logical-mathematical description in four levels of the design object model. At each level, modeled and refined objects set their links (interfaces) and relationships, the basic characteristics of logic functions and the behavior of objects. Objects and their interfaces are grouped into classes, and are displaying in the graph object model. Objects are transformed to the form of components maintaining relationships. The formal model (the component, interface, environment, and system) of the component environment is proposed, technical transfer of the objects into the component environment and assembly of objects and components into the system configuration file to execute is provided.

Keywords: modeling, logical-mathematical apparatus, subject area, level simulation, models, objects, graph components, variability, reusable components, assembly, configuration

References

[1]. Lavrischeva E.M., Grishenko V.Y. Metody i sredstva ob#ektno-komponentnogo programmirovanija [Methods and tools for object-component programming] // Kibernetika i sistemnyj analiz [Cybernetics and Systems Analysis], 2003.– №1.– pp. 39–55 (in Russian).
[2]. Lavrischeva E.M., Grishenko V.Y. Svjaz' raznojazykovyh modulej v OS ES [Communication of multilingual modules in OS ES] //M.: Finansy i statistika [Finance and Statistics], 1982.–127 p. (in Russian).
[3]. Andon Ph.I., Lavrischeva E.M. Metody inzhenerii komp'juternyh prilozhenij. Ob'ektnyj podhod [Methods of engineering computer applications. An object approach // K: Naukova Dumka [Scientific Thought], 1997.–229 p. (in Russian).
[4]. Grishenko V.N. Metod ob'ektno-komponentnogo proektirovanija programmnyh sistem [A method for object-component design of software systems] // Problemy programmirovanija [The Problems of Programming]. – 2007. – №2. – pp.113–125. (in Russian).
[5]. Grishenko V.N. Teoreticheskie i prikladnye aspekty komponentnogo programmirovanija [Theoretical and applied aspects of component programming] // Avtoreferat dokt. Dissertacii [Abstract of the Doctoral Thesus], Kiev, Glushkov Institute of Cybernetics of National Academy of Sciences of Ukraine.–2007.–34 p. (in Russian).
[6]. Lavrischeva E.M. Metody programmirovanija. Teorija, inzhenerija, praktika [Programming Methods. Theory, engineering, practice] // K: Naukova Dumka [Scientific Thought], 2006.– 451 p. (www.twirpx.com) (in Russian).

[7]. Lavrischeva E.M., Grishenko V.N. Sborochnoe programmirovanie [Assempling Programming]. – K: Naukova Dumka [Scientific Thought], 1991.-216 p. (in Russian).
[8]. Lavrischeva E.M., Grishenko V.N. Sborochnoe programmirovanie. Osnovy industrii programmnyh produktov [Assempling Programming. Fundamentals of software industry]. K: Naukova Dumka [Scientific Thought], 2009.–371 p. (in Russian).
[9]. Lavrischeva E.M., Kolesnik A.D., Koval' G.I., Slabospitskaya O.A. Teoreticheskie aspekty upravlenija variabel'nost'ju v semejstvah programmnyh sistem [Theoretical aspects of managing variability in families of software systems]. – Bulletin of Kyiv National Taras Shevchenko University, series of physical and mathematical sciences. – 2011, №1, pp.151 –158 (in Ukrainian).
[10]. Lavrischeva E.M., Kolesnik A.D., Stenyashin A.Yu. Ob'ektno-komponentnoe proektirovanie programmnyh sistem. Teoreticheskie i prikladnye voprosy [Object-component design of software systems. Theoretical and Applied Problems]. Bulletin of Kyiv National Taras Shevchenko University, series of physical and mathematical sciences. – 2013. – №4. – pp. 150 – 164 (in Ukrainian).
[11]. Slabospitskaya O.A. Modeli i metody jekspertnogo ocenivanija v ZhC PS [Models and methods for expert evaluation in the life cycle of software] // Avtoref. kand. dis. [Abstract of the PhD Thesis. – Kiev, Glushkov Institute of Cybernetics of National Academy of Sciences of Ukraine.– 2008.– 22 p. (in Russian).
[12]. Ekaterina Lavrischeva, Andrey Stenyashin, Andriy Kolesnyk, Object-Component Development of Application and Systems. Theory and Practice, Journal of Software Engineering and Applications, 2014, 7, Published Online August 2014 in SciRes http://www.scirp.org/journal/jsea
[13]. Lavrischeva K.M., Slabospitskaya O.O. Pidhid do pobudovy ob'jektno-komponentnoi' modeli simejstva programnyh produktiv [The approach to building component object model for a family of software products] // Problemы programuvannja [The Problems of Programming], 2013.–№3.– pp. 14–26. (in Ukrainian).
[14]. Lavrischeva K.M., Zin'kovitch V.M.. Kolesnik A.L. et al. Instrumental'no-tehnologicheskij kompleks razrabotki i obuchenija priemam izgotovlenija programmnyh sistem [Instrumental and technological complex for development and training methods of manufacturing software systems].– Gos. sluzhba intellektual'noj sobstvennosti Ukrainy.– Svidetel'stvo o registracii avtorskogo prava [The State Service of Intellectual Property of Ukraine.- Certificate of copyright registration]. – № 45292, 27.08.2012.
[15]. Lavrischeva K.M. Software Engineering komp'juternyh sistem. Paradigmy, Tehnologii, CASE-sredstva programmirovanija [Software Engineering of Computer Systems. Paradigms, Technologies, Programming CASE-tools. – K: Naukova Dumka [Scientific Thought], 2014. –284 p. (in Russian).
[16]. Kolesnyk A., Slabospitskaya O. Tested Approach for Variability Management Enhancing in Software Product Line. – In: ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge, Proc. 8 –th Int. Conf. ICTERI 2012, Kherson, Ukraine, June 6 –10, 2012, CEUR –WS.org/Vol –848, ISSN 1613 –0073, urn:nbn:de:0074 –848-8. – P. 125 –133.
[17]. Kolesnyk A.L. Modeli i metody razrabotki semejstva variantnyh programmnyh sistem [Models and methods for the development of a family of variant software systems]. – Avtoref. kand. dis. [Abstract of the PhD Thesis. – Kiev, Glushkov Institute of Cybernetics of National Academy of Sciences of Ukraine, 2013. –22c. (in Ukrainian).
[18]. Lavrischeva K., Aronov A., Dzubenko A. Programs Factory – A conception of Knowledge Reprezentation of Scientifical Artifacts From Standpoint of Software Engineering.– Computer and Information Science, Canadion Center of Science and Education.–2013.–ISSN 1913 –8989 (Print).–P.21–28.
[19]. Lavrischeva K.M. Komponentnaja teorija i kollekcija tehnologij razrabotki industrial'nyh prilozhenij iz gotovyh resursov [Component theory and collection of technologies for development of industrial applications of ready resources, Trudy Chetvertoj nauchnoj konferencii «Aktual'nye problemy sistemnoj i programmnoj inzhenerii» [Proceedings of the Fourth scientific conference "Actual problems of system and software engineering"], May 20 –21, 2015, pp. 101 –119. (in Russian).
[20]. Ostrovski A.I. Podhod k obespecheniju vzaimodejstvija programmnyh sred JAVA i Ms.Net [The approach to the interoperability Ms.Net and JAVA programming environments – Problemy programmirovanija [The Problems of Programming], 2011.–№ 2.– pp. 37–44. (in Russian).
[21]. Radetski I.O. Odin iz podhodov obespechenija vzaimodejstvija sred MS.Net i Eclipse [One approach to the interoperability of MS.Net and Eclipse environments / Problemy programmirovanija [The Problems of Programming], № 2, 2011.– pp. 45–52 (in Russian).