УДК 004.435

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ МЕТАЯЗЫК

Н. А. Тюкачев, А. А. Седунов

Воронежский государственный университет

Поступила в редакцию 1.03.2009 г.

Аннотация. В рамках данной работы рассматривается оригинальный подход к решению проблемы разработки метаязыка путем соединения логического и объектно-ориентированного подходов, приводится описание типов и операций языка, вводится определение трансформационной семантики как набора правил преобразования модели предметной области в формальную теорию над исчислением предикатов первого порядка с равенством

Ключевые слова метаязык, ООП, семантика, исчисление предикатов, логическое программирование

Abstract. In this paper the authors consider original method of solving the specification metalanguage design problem by composing logical and object-oriented paradigms. Language types and operations are presented along with syntax details description and definition of transformational semantics in terms of translation from domain model to formal theory over the first-order predicate calculus with equality.

Keywords: metalanguage, OOP, semantics, predicate calculus, logic programming

ВВЕДЕНИЕ

Разработка программного обеспечения [1, 3] в настоящее время сопряжена с необходимостью составления и поддержки различного рода документации, качество и полнота которой во многом определяет качество и надежность программной системы, а также возможность ее модификации и расширения в случае изменения требований. Однако, как показывает практика, даже наличие полноценной документации не позволяет гарантировать надежность и корректность работы сложного программного продукта. Поведение реальных программных систем всегда отличается от исходных требований и приводит к возникновению незапланированных эффектов. Причиной данного явления является высокая сложность современных программных систем, которая может выражаться в формулировке функциональных требований, анализе вариантов использования системы и взаимодействия с ее окружением, а также в необходимости разработки системы с учетом возможных изменений в требованиях. Как правило, информация о системе закрепляется в документах, составленных с использованием естественного языка, либо частично формализованной нотации (например, UML [2]). Автоматизация ее анализа и применения в большинстве CASE-средств существенно ограничена, либо вообще невозможна. Более того, такое описание часто не отличается точностью и является неоднозначным. Принципиальное решение проблемы состоит в автоматизации процесса разработки и изменении подхода к документированию информации о разрабатываемом программном продукте.

Формальные методы разработки ПО представляют собой [1, 5] особую разновидность математических моделей, предназначенных для составления спецификаций ПО, построения программного кода, а также проверки корректности программ. Главное преимущество формального подхода состоит в возможности применения математических методов к разработке и анализу программного обеспечения, позволяя тем самым гарантировать его корректность и надежность. Ядром инструмента автоматизации разработки является специальный метаязык, с помощью которого разработчик может описывать предметную область решаемой проблемы [4, 5]. В данной работе предлагается при построении метаязыка объединить формализм логики предикатов и элементы теории множеств с объектно-ориентированным подходом

Исследование, выполненное авторами в рамках проекта JANE по созданию инструментальной среды для разработки и анализа фор-

мальных спецификаций, представляет собой оригинальный способ построения гибридного метаязыка. В данной статье описывается часть языка JLS, касающаяся структуры и семантики типов и операций без рассмотрения выражений (термов и предикатов) языка.

1. ЯЗЫК МОДЕЛИРОВАНИЯ JLS. JLS-МОДЕЛИ

JLS-модель описывает набор взаимосвязанных сущностей предметной области. Каждая модель образует отдельное пространство имен, состоящее из определений классов. Возможно также использование определений из других моделей – в этом случае имеет место зависимость между моделью, использующей объект и моделью, содержащей его определение. Внешние определения должны квалифицироваться именем модели. Граф отношения зависимости между моделями не должен содержать циклов – в противном случае все модели, образующих цикл, считаются некорректными.

Процедура трансформации получает на входе корректную JLS-модель и строит на выходе соответствующую ей формальную теорию над исчислением предикатов первого порядка с равенством, которая может использоваться алгоритмами логического вывода для получения ответов вопросы в рамках предметной области модели (включая проверку истинности утверждений, поиск решений, синтез программ и т. д.).

Формальная теория, построенная в результате трансформации модели, характеризуется множествами предикатных и функциональных символов, набор собственных аксиом, устанавливающих свойства предикатов и функций, а также множеством утверждений V, которые должны быть верифицированы для подтверждения корректности модели. Если хотя бы одно из утверждений V не удается проверить, модель считается некорректной.

1.2.ТИПЫ И ОПЕРАЦИИ

1.1.1. СИСТЕМА ТИПОВ JLS

Тип характеризует множество значений, которые могут принимать объекты модели, а также допустимые операции и отношения, в которых они могут использоваться. В JLS типы существуют только на уровне модели и при трансформации преобразуются в предикаты соответствующей формальной теории.

Система типов вводится на основе иерархически определяемых множеств. Пусть $\mathbf{T}_0$ - некоторое множество, далее именуемое множеством *типов нулевого уровня* (*элементарных типов*). Множество *типов уровня* n, где $n > 0$ рекуррентно определяется следующим образом:

$$\mathbf{T}_n \overset{def}{=} \mathbb{P}[\mathbf{T}_{n-1}] \overset{def}{=} \{\mathbb{P}(T) \mid T \in \mathbf{T}_{n-1}\}, n > 0, \quad (1)$$

где $\mathbb{P}$ - оператор, называемый *оператором степенного типа*.

Тип $T = \mathbb{P}(S)$ называется *степенным типом с базовым типом* S.

Системой типов JLS называется множество

$$\mathbf{T} = \mathbf{T}_\infty \overset{def}{=} \bigcup_{n \geq 0} \mathbf{T}_n. \quad (2)$$

Оператор степенного типа представляет собой биективное отображение вида $\mathbb{P} : \mathbf{T} \to \mathbf{T}^+$, характеризуемое следующим свойством:

$$\forall (n \geq 0) \forall (T \in \mathbf{T}_n) [\mathbb{P}(T) \in \mathbf{T}_{n+1}]. \quad (3)$$

Два типа T и S называются *подобными*, если они находятся на одном уровне системы типов:

$$T \sim S \overset{def}{=} \exists (n \geq 0) [T \in \mathbf{T}_n \wedge S \in \mathbf{T}_n]. \quad (4)$$

Каждый тип $T \in \mathbf{T}$ характеризуется множеством своих экземпляров, которое будет обозначаться через $Set[T]$. Если $T = \mathbb{P}(S)$ - степенной тип, то множество его экземпляров определяется как степенное множество по отношению к множеству экземпляров базового типа S:

$$Set[\mathbb{P}(S)] \overset{def}{=} 2^{Set[S]}. \quad (5)$$

Константный тип Any представляет собой корневой тип нулевого уровня, т. е. любой элементарный тип является подтипом Any. Введение Any (как и других корневых типов) в систему типов JLS связано с необходимостью поддержки обобщенных операций, тип параметров которых не всегда может быть известен заранее.

Множеством экземпляров типа Any является специальная предметная константа Any^*, которая автоматически включается в формальную теорию модели в процессе трансформации.

$$Set[Any] \overset{def}{=} Any^*. \quad (6)$$

Состав множества $Any*$ не определяется явным образом, однако аксиомы, генерируемые при трансформации модели, гарантируют, что

множество экземпляров любого элементарного типа будет подмножеством Any^*.

Основным отношением, определенным в системе типов JLS, является отношение специализации $<:$, выражающее тот факт, что экземпляр одного (*подтипа*) всегда может рассматриваться как экземпляр другого типа (*супертипа*). Отношение специализации характеризуется рядом свойств:

$$\begin{cases} T <: T \\ T <: S, S <: R \Rightarrow T <: R \\ T <: S \Rightarrow Set[T] \subseteq Set[S] \\ T <: S \Rightarrow Op[S] \subseteq Op[T] \end{cases} \quad (7)$$

Точное определение специализации в случае элементарных типов зависит от их структуры. Для степенных типов специализация определяется следующим образом. Пусть T и S - подобные типы, тогда тип T является подтипом типа S тогда и только тогда, когда базовый тип T является подтипом базового типа S:

$$T <: S \overset{def}{\Leftrightarrow} T \sim S \wedge \mathbb{P}^{-1}(T) <: \mathbb{P}^{-1}(S). \quad (8)$$

Два подобных типа T и S называются *эквивалентными*, если они являются подтипами друг друга:

$$T \approx S \overset{def}{\Leftrightarrow} T <: S \wedge S <: T. \quad (9)$$

В языке JLS эквивалентные типы фактически являются неразличимыми, так как любой из них может использовать в контексте, требующем второго типа.

1.1.2. ОПЕРАЦИИ

Каждый элементарный тип T характеризуется множеством $Op[T]$ операций, которые могут быть выполнены над его экземплярами. В JLS существуют две разновидности операций: отношения и функции.

Каждая операция r характеризуется следующими свойствами:

Логический признак абстрактной операции A. Операция, имеющая этот признак ($A = 1$), называется *абстрактной*, не имеющая ($A = 0$) – *замкнутой*.

Имя операции $r.N$ – произвольная текстовая строка, идентифицирующая данную операцию среди множества всех операций соответствующего типа. Все операции, принадлежащие одному типу, должны иметь различные имена.

Кортеж параметров $r.P = \langle P_i \rangle_{i=1}^{m_r}$. Каждый параметр P_i характеризуется своим *типом* $P_i.T$ и необязательным предикатом *ограничения* $P_i.C = P_i.C(p_i)$. Если C_i не задан, подразумевается тождественно-истинное утверждение.

Необязательный предикат $R = R(p_1, ..., p_m)$, называемый *предусловием* и используемый для дополнительного ограничения области изменения значений параметров. Если предусловие отсутствует, подразумевается тождественно-истинное утверждение.

Тело операции, которое представляет собой выражение вида $B = B(t, p_1, ..., p_m)$. Тело операции может зависеть от ее параметров, а также от this-переменной t, представляющей произвольный экземпляр типа, которому принадлежит данная операция. Для операции-отношения выражение B является предикатом, для операции-функции – термом. Для абстрактной операции тело не определено.

Сигнатурой операции r называется кортеж $sig(r)$:

$$sig(r) = \langle K, N, P \rangle, \quad (10)$$

где K - вид операции (для отношения $K = 0$, для функции $K = 1$).

Полной сигнатурой $Sig(r)$ операции называется кортеж вида

$$Sig(r) = \begin{cases} sig(r), sig(r).K = 0 \\ \langle sig(r), T_0 \rangle, sig(r).K = 1 \end{cases} \quad (11)$$

Операции-функции дополнительно характеризуются типом результата T_0 и необязательными ограничениями значения функции $C_0 = C_0(s)$ и $E = E(p_1, ..., p_m, s)$, которые называется *result-ограничением* и *постусловием* соответственно.

1.1.3. КЛАССЫ

СТРУКТУРА КЛАССА

Классы представляют собой основную разновидность типов в языке JLS, предназначенную для описания сущностей предметной области модели. Классы позволяют непосредственно определять набор операций, доступных для своих экземпляров и специализацию по отношению к другим типам. Все остальные элементарные типы, кроме Any, строятся на основе классов.

Класс C характеризуется следующим набором свойств:

· Набор атрибутов

$$C.A \subseteq \begin{Bmatrix} Primal, Abstract, Unique, \\ Closed, Disjoint, Case, Final \end{Bmatrix}.$$

· *Имя класса* $C.N$ – произвольная текстовая строка, идентифицирующая данный класс

среди других классов модели. В пределах одной модели все классы должны иметь различные имена.

· *Кортеж параметров* $C.P = \langle P_i \rangle_{i=1}^{m}$ и предусловие $C.R = C.R(p_1, ..., p_m)$. Эти свойства аналогичны свойствам функций и отношений и определяют условия существования экземпляра класса.

· *Определение класса* $C.D = C.D(p_1, ..., p_m)$. Определение – терм элементарного типа, который может зависеть от параметров класса, а также его произвольного экземпляра t (this-переменная). Тип определения называется *прямым супертипом* класса C. Определение может отсутствовать – в этом случае прямым супертипом класса является Any.

· Произвольный предикат $C.I = C.I(t, p_1, ..., p_m)$, который называется *инвариантом* и может зависеть от параметров класса и this-переменной. Инвариант выражает условие, которое выполняется для любого экземпляра класса.

· Множество Op_{imm} *непосредственных операций*. Выражения, входящие в эти операции, могут зависеть от параметров класса и this-переменной.

Атрибуты классов

Поскольку утверждения о существовании объектов с теми или иными свойствами часто применяются при построении математических моделей, в языке JLS вводится специальные соглашения, упрощающие использование таких утверждений.

Атрибут Primal связан с проверкой существования экземпляров класса. В JLS каждый класс должен иметь хотя бы один экземпляр, т. е. утверждение, описывающее принадлежность произвольного объекта данному классу должно быть выполнимым в логике предикатов. Если класс отмечен атрибутом Primal (такой класс называется *первичным*), то соответствующее утверждение о существовании будет добавлено к списку аксиом формальной теории и не будет верифицировано в процессе трансформации. В противном случае система попытается вывести это утверждение из аксиом теории. Если выполнить вывод не удается, модель считается некорректной.

Смысл автоматически генерируемого утверждения можно неформально выразить следующим образом: для любой комбинации объектов, удовлетворяющих ограничениям класса, существует хотя бы один экземпляр, значения параметров которого совпадают с этими объектами.

Другая важная категория утверждений, также часто применяемых в математических моделях, описывают единственность объекта, характеризуемого заданными свойствами. В языке JLS для поддержки утверждений этого типа вводятся атрибуты класса Abstract и Unique. В зависимости от наличия или отсутствия этих атрибутов все классы делятся на 3 группы:

· Если класс отмечен атрибутом Abstract, он называется *абстрактным*.

· Если класс не отмечен ни одним из атрибутов Abstract или Unique, он называется *неявно уникальным*.

· Если класс отмечен атрибутом Unique, он называется *явно уникальным*.

Одновременное использование атрибутов Abstract и Unique не разрешается.

Смысл утверждения о единственности можно неформально выразить следующим образом: если для любой комбинации объектов, удовлетворяющих ограничениям параметров конструктора, существует хотя бы один экземпляр класса, значения параметров которого совпадают с этими объектами, то такой экземпляр является единственным.

Если класс является абстрактным, то утверждение о единственности не генерируется и не проверяется – тем самым подразумевается, что единственность экземпляра не гарантируется. Класс, содержащий хотя бы один абстрактный метод, должен обязательно быть отмечен атрибутом Abstract (в силу того, что абстрактный метод может иметь различные реализации в подклассах, гарантировать существование только одного экземпляра для такого класса нельзя и, следовательно, он обязательно должен быть абстрактным).

Для неявно уникального класса утверждение о единственности генерируется и в процессе трансформации модели выводится из имеющихся аксиом теории. Если вывод не удается, модель считается некорректной.

В случае с явно уникальным классом утверждение о единственности генерируется, но не верифицируется, а включается в число аксиом теории.

Класс, отмеченный атрибутом Closed, называется *замкнутым*. Замкнутый класс позволя-

ет описать специальный набор Case-подклассов, характеризуемый следующим свойством: объединение множеств экземпляров Case-подклассов совпадает с множеством экземпляров данного класса. Case-подклассы отмечаются с помощью атрибута Case и должны принадлежать той же модели, что и данный класс. При этом замкнутый класс, как и любой другой, может иметь обычные (не Case) подклассы.

Замкнутый класс, отмеченный атрибутом Disjoint, называется *дизъюнктным*. Дизъюнктный класс характеризуется тем, что если некоторый объект является экземпляром одного из его Case-подклассов, то он не может быть экземпляром никакого другого его Caseподкласса (иначе говоря, множества экземпляров Case-подклассов попарно не пересекаются). Дизъюнктный класс автоматически становится замкнутым, независимо от наличия атрибута Closed.

Класс, отмеченный атрибутом Final, называется *завершенным*. Такой класс не может использоваться в качестве суперкласса (непосредственно или в составе пересечения). Все методы завершенного класса автоматически становятся замкнутыми.

Экземпляры и операции класса

Пусть $C = \langle A, N, P, R, D, I, Op_{imm} \rangle$ - класс с прямым супертипом T.

Множество экземпляров класса определяется с помощью вспомогательного предиката $\mathrm{is}C(t, p_1, ..., p_m)$ (IS-предикат), который выражает тот факт, что объект t является экземпляром класса C со значениями параметров $p_1, ...p_m$. Формальное определение IS-предиката будет дано при описании трансформационной семантики классов.

$$Set[C] = \\ = \left\{ t \in T \,\middle|\, \exists (p_1, ...p_m) \left[\mathrm{is}C\,(t, p_1, ..., p_m) \right] \right\}. \quad (12)$$

Для определения множества операций $Op[C]$ необходимо ввести некоторые вспомогательные понятия. Пусть $r \in Op_{imm}$ - непосредственная операция C. Если существует такая операция $r' \in Op[T]$, что $r'.N = r.N$, то r называется *потенциально переопределяющей операцией* (по отношению к r'):

$$r \underset{?}{\leftarrow} r' \overset{def}{\Leftrightarrow} r.N = r'.N. \quad (13)$$

Если при этом полные сигнатуры r и r' совпадают, r не является абстрактной, не содержит явных ограничений параметров и не определяет явным образом предусловие, то операция r называется *переопределяющей операцией* (также по отношению к r'):

$$r \leftarrow r' \overset{def}{\Leftrightarrow} r \underset{?}{\leftarrow} r' \wedge Sig(r) = \\ = Sig(r') \wedge r.A = 0 \wedge \\ \wedge r.R(x_1, ..., x_k) \equiv \\ \equiv True \wedge \left(\bigwedge_{i=1}^{k} r.P_i.C \equiv True \right). \quad (14)$$

В противном случае r и r' называются *конфликтующими операциями*. Пусть теперь

· Op_{imp} - множество *неявных операций* прямого супертипа, которые потенциально не переопределяются какой-либо операцией Op_{imm}:

$$Op_{imp} = \\ = \{ r' \in Op[T] \mid \neg\exists (r \in Op_{imm})[r \underset{?}{\leftarrow} r'] \mid \}. \quad (15)$$

· Op_{pro} - множество *собственных операций класса* C. Собственная операция – это непосредственная операция, которая не является потенциально переопределяющей по отношению к какой-либо операции из набора $Op[T]$:

$$Op_{pro} = \\ = \left\{ r \in Op_{imm} \,\middle|\, \neg\exists \left(r' \in Op[T] \right) \left[r \underset{?}{\leftarrow} r' \right] \right\}. \quad (16)$$

· Op_{ovr} - множество операций Op_{imm}, *переопределяющих операции* $Op[T]$:

$$Op_{ovr} = \\ = \left\{ r \in Op_{imm} \,\middle|\, \exists \left(r' \in Op[T] \right) \left[r \leftarrow r' \right] \right\}. \quad (17)$$

· Op_{cnf} - множество операций, *конфликтующих с операциями* $Op[T]$:

$$Op_{cnf} = Op[T] \cap Op_{imm} \setminus Op_{ovr}. \quad (18)$$

Если конфликтующих операций нет ($Op_{cnf} = \varnothing$), то класс считается корректным и набор его операций выражается через введенные множества:

$$Op[C] \overset{def}{=} Op_{imp} \cup Op_{pro} \cup Op_{ovr}. \quad (19)$$

В противном случае ($Op_{cnf} \neq \varnothing$) класс считается некорректным и множество $Op[C]$ не определено.

Правило специализации для классов определяется следующим образом. Пусть C - класс с прямым супертипом T и S - элементарный тип, тогда:

$$C <: S \overset{def}{\Leftrightarrow} T <: S. \quad (20)$$

1.1.1. ПРОИЗВОДНЫЕ ТИПЫ

Категория производных типов индуктивно определяется следующим образом. Пусть T - класс или ранее построенный производный тип с непустым множеством операций, $r \in Op[T]$ - одна из его операций с именем $x = r.N$ и y - произвольное имя, не совпадающее ни с одним из имен операций $Op[T]$. Тогда выражение $T[x = y]$ называется *производным типом, образованным от* T *подстановкой* $x := y$ (тип T при этом называется *исходным типом*).

Из определения следует, что любой производный тип T можно представить в виде

$$T = S[x_1 = y_1]...[x_k = y_k], \qquad (21)$$

где $k > 0$. В представлении тип S называется *основным исходным* типом T.

Множество значений производного типа совпадает с множеством значений соответствующего исходного типа:

$$Set\big[T[x = y]\big] \overset{def}{=} Set[T]. \qquad (22)$$

Множество операций производного типа фактически совпадает с множеством операций его исходного типа за исключением того, что операция r меняет имя с x на y. Эта операция называется *производной операцией* типа $T[x = y]$, все остальные операции называются *неявными* (*неявно унаследованными* от исходного типа).

$$Op\big[T[x = y]\big] \overset{def}{=} Op[T] \setminus \{r\} \cup \cup \big\{r_{N=y}\big\}, r.N = x, \qquad (23)$$

где $r_{N=y}$ обозначает операцию, все атрибуты которой (кроме имени N) совпадают с соответствующими атрибутами операции r, а имя совпадает с y.

Таким образом, эффект производного типа состоит в простом переименовании одной из операций исходного типа без изменения ее семантики. Данная операция является необходимой во многих случаях множественного (в частности, повторного) наследования классов.

Правила специализации для производных типов сводятся к следующему: $T[x = y] <: S$ тогда и только тогда, когда S имеет вид $U[x = y]$, где $T <: U$:

$$T[x = y] <: S \overset{def}{\Leftrightarrow} S = U[x = y] \wedge T <: U. \qquad (24)$$

Из правила и данного выше определения $T[x = y]$ следует, что специализация производных типов сводится к специализации классов, на основе которых они построены.

Производные типы не связаны с соответствующими исходными типами отношением специализации. В частности, терм, имеющий тип $T[x = y]$ не может непосредственно использоваться в контексте, требующем объект типа T.

1.1.2. ТИПЫ-ПЕРЕСЕЧЕНИЯ

Типы-пересечения используются в языке JLS, главным образом, для реализации множественного наследования при описании классов. Для определения понятия типа-пересечения вначале необходимо рассмотреть ряд вспомогательных понятий.

Пусть $\{T_i\}_{i=1}^m$ - множество типов, в котором каждый тип является либо производным, либо классом, $Op_i = Op[T_i]$ - множество операций i-го типа и $Op = \bigcup_{i=1}^{m} Op_i$. Следующая процедура строит на основе множества Op три множества Op_{imp}, Op_{mrg} и Op_{cnf}, которые называются соответственно множествами *неявных*, *объединенных* и *конфликтующих* операций для набора типов $\{T_i\}_{i=1}^m$:

1. Если множество Op пусто, завершить процедуру.

2. Пусть $r \in Op$ - произвольная операция из набора Op и i_0 - номер исходного набора, которому она принадлежит, т. е. $r \in Op_{i_0} \subseteq Op$. Если ни в одном из наборов, отличных от Op_{i_0} не содержится операции, имя которой совпадает с именем r, то операция r добавляется к множеству Op_{imp} и удаляется из множества Op. Затем происходит переход к шагу 1.

$$\begin{cases} Op \leftarrow Op \setminus \{r\} \\ Op_{imp} \leftarrow Op_{imp} \cup \{r\} \end{cases} \qquad (25)$$

3. Пусть $R = \big\{\langle r_i, T_i\rangle \big| r_i \in Op_i \wedge r_i.N = r.N\big\}$ - множество пар, включающих операции, имена которых совпадают с $r.N$ (в частности, $r \in R$) вместе с соответствующими типами из набора $\{T_i\}_{i=1}^m$. Если множество R является группой родства (определение соответствующему понятию дается в параграфе "Родственные операции"), то все операции, принадлежащие парам R, удаляются из Op, а к множеству Op_{mrg} добавляется операция $merge(R)$. После этого происходит переход к шагу 1:

$$\begin{cases} R' = \big\{r' \big| \exists(i)\big[\langle r', T_i\rangle \in R\big]\big\} \\ Op \leftarrow Op \setminus R' \\ Op_{imp} \leftarrow Op_{imp} \cup \{merge(R)\} \end{cases} \qquad (26)$$

4. Все операции, принадлежащие парам R, удаляются из Op и добавляются к множеству Op_{cnf}:

$$\begin{cases} Op \leftarrow Op \setminus R' \\ Op_{imp} \leftarrow Op_{imp} \cup R' \end{cases} \quad (27)$$

Если множество R является группой родства, то все операции в R' имеют одинаковые полные сигнатуры (такую же полную сигнатуру имеет и операция $merge(R)$). Если все операции в R' имеют вид $r_j = \langle A_j, N_0, P_j, R_j, B_j \rangle, j = 1..l$, тогда

$$merge(R) \stackrel{def}{=} \langle A_0, N_0, P_0, R_0, B_0 \rangle, \quad (28)$$

где

$$\begin{cases} A_0 = \min\limits_{j=1..l} A_j \\ P_0 = \langle P_i \rangle, P_i.C(t) = \bigwedge\limits_{j=1}^{l} r_j.P_i.C(t) \\ R_0(p_1,...,p_n) = \bigwedge\limits_{j=1}^{l} r_j.R(p_1,...,p_n) \end{cases} \quad (29)$$

и, если $A_0 = 0$, а j_0 - номер неабстрактной операции (в группе родства такая операция, если вообще существует, является единственной):

$$B_0(p_1,...,p_n,t) = B_{j_0}(p_1,...,p_n,t). \quad (30)$$

Если $A_0 = 1$, то B_0 не определено.

Таким образом, объединенная операция $merge(R)$ является абстрактной, если все операции, входящие в R', являются абстрактными и замкнутой – в противном случае. Ограничения параметров и предусловие объединенной операции представляют собой конъюнкции соответствующих предикатов операций R'. Тело объединенной операции (если она не является абстрактной) совпадает с телом замкнутой операции из R'.

Если $Op_{cnf} = \varnothing$ (т. е. конфликтующих операций нет), то $T = \bigcap\limits_{i=1}^{m} T_i$ называется типом-пересечением, образованным типами $T_i, i = 1..m$. При этом

$$Set\left[\bigcap_{i=1}^{m} T_i\right] \stackrel{def}{=} \bigcap_{i=1}^{m} Set\left[T_i\right] \quad (31)$$

и

$$Op\left[\bigcap_{i=1}^{m} T_i\right] \stackrel{def}{=} Op_{imp} \cup Op_{mrg} \quad (32)$$

Если типы, образующие пересечение, имеют одноименные, но не связанные друг с другом операции, множество Op_{cnf} окажется непустым, а пересечение будет не определено. В этом случае один или несколько типов, содержащих конфликтующие операции, следует заменить на производные типы, в которых этим операциям присваиваются различные (неконфликтующие) имена.

Для типов-пересечений вводится два правила специализации. Тип $T = \bigcap\limits_{i=1}^{m} T_i$ называется подтипом типа S, если

· Тип S является производным типом или классом, причем

$$\exists(i = 1..n)[T_i <: S]. \quad (33)$$

· Тип S является пересечением $S = \bigcap\limits_{i=1}^{k} S_i$, причем

$$\forall(i = 1..k)\exists(j = 1..m)[T_j <: S_i]. \quad (34)$$

1.1.1. РОДСТВЕННЫЕ ОПЕРАЦИИ

На основе отношения специализации вводится ряд отношений, связывающих операции, принадлежащие типам. Эти отношения совместно с выражениями формального вызова позволяют формализовать полиморфное поведение операций.

Пусть $T <: S$, $r_T \in Op[T]$ и $r_S \in Op[S]$ - две операции с одинаковой полной сигнатурой, рассматриваемые в контексте типов T и S соответственно и пусть, по крайней мере, одна из них является абстрактной. Операция r_S называется *базовой* по отношению к операции r_T (запись: $r_T <: r_S$), если выполняется одно из следующих условий:

· $r_T = r_S$;

· Существует класс U и операция $r_U \in Op[U]$, такие, что $T <: U, U <: S$ и $r_T <: r_U, r_U <: r_S$;

· T - класс с прямым супертипом U и r_T - операция, переопределяющая операцию $r_U \in Op[U]$, причем $r_U <: r_S$.

· $T = \bigcap\limits_{i=1}^{m} T_i$, $r_T = merge(R), R \subseteq Op$ и $r_S \in R$.

Операция r_S называется *корневой* по отношению к операции r_T (запись: $r_T <<: r_S$), если $r_T <: r_S$ и не существует базовой операции для r_S.

Две операции r_T и r_S называются *родственными* в контексте типов T и S, если они имеют общую базовую операцию. Множество $R = \left\{\langle r_{T_i}, T_i \rangle\right\}_{i=1}^{m}$, где $r_{T_i} \in Op[T_i]$ называется *группой родства*, если любые две операции r_{T_i} и r_{T_j},

входящие в него, родственны в контексте соответствующих типов T_i и T_j.

1.2. СЕМАНТИКА ТИПОВ И ОПЕРАЦИЙ

1.2.1. КЛАССЫ

Пусть C - класс в модели M. Тогда в теорию F_M включается предметная константа C^*, соответствующая множеству его экземпляров $Set[C]$, и $(m+1)$-арный предикат $\mathrm{is}C(t, x_1, ..., x_m)$, где m - количество параметров конструктора класса C. Неформально смысл этого предиката заключается в том, что объект t является экземпляром класса C с параметрами $x_1, ..., x_m$.

Константа C^* вводится с помощью упрощенного определения:

$$\forall (t)[t \in C^* \leftrightarrow \leftrightarrow \exists (p_1, ...p_m)[\mathrm{is}C(t, p_1, ..., p_m)]]. \quad (35)$$

Предикат $\mathrm{is}C$ называется *is-предикатом*. Для его описания вначале вводится вспомогательное понятие *фактического предусловия* класса C, которое представляет собой предикат $R_C^* = R_C^*(x_1, ..., x_m)$:

$$R_C^*(x_1, ..., x_m) \overset{def}{=} \bigwedge_{i=1}^{m} (\llbracket x_i : T_i \rrbracket \wedge \llbracket C_i(x_i) \rrbracket) \wedge \wedge \llbracket R(x_1, ..., x_m) \rrbracket. \quad (36)$$

is-предикат определяется следующим образом:

$$\mathrm{is}C(t, x_1, ..., x_m) \overset{def}{=} R_C^*(x_1, ..., x_m) \wedge \wedge \llbracket C.D(x_1, ..., x_m) \rrbracket_t \wedge \llbracket I(t, x_1, ...x_m) \rrbracket. \quad (37)$$

Выражение $\llbracket C.D(x_1, ..., x_m) \rrbracket_t$ представляет собой предикат, зависящий от параметров $x_1, ...x_m$ и переменной t и означающий, что объект t имеет значение терма $\llbracket C.D(x_1, ..., x_m) \rrbracket$. Для однозначных термов этот предикат сводится к равенству, но в общем случае его определение усложняется, так как JLS разрешает использовать многозначные термы в определениях классов. Фактическое определение $\llbracket C.D(x_1, ..., x_m) \rrbracket_t$ зависит от структуры термов и здесь не рассматривается.

Для каждого параметра класса генерируется специальная унарная *функция-селектор* $\mathrm{par}C.x(t)$, где x - это имя параметра. Для любого экземпляра класса t выражение $\mathrm{par}C.x(t)$ соответствует значению параметра x, которое связано с этим экземпляром. Формально селектор определяется с помощью аксиомы:

$$\forall (t)\forall (p) [\mathrm{par}C.x(t) = p \leftrightarrow \leftrightarrow \exists (y_1, ...y_{i-1}, y_{i+1}, ...y_m) \times \times [isC(t, y_1, ...y_{i-1}, p, y_{i+1}, ...y_m)]], \quad (38)$$

где i - это номер параметра с именем x.

1.2.2. ПРЕДИКАТЫ ТИПОВ

При трансформации модели информация о типах сохраняется в виде специальных предикатов, которые называются *предикатами типов*. Предикаты типов будут обозначаться $\llbracket t : T \rrbracket$, где t - **JLS-терм, а** T - тип.

Семантика определяется в зависимости от структуры типа T и сводится к следующему набору правил:

1. $$\llbracket t : \mathbb{P}(U) \rrbracket \overset{def}{=} \forall (x)[x \in \llbracket t \rrbracket \rightarrow \llbracket x : U \rrbracket], \quad (39)$$

где x - это имя, не встречающееся в терме t.

2. $$\llbracket t : Any \rrbracket \overset{def}{=} \llbracket t \rrbracket \in \mathrm{U}. \quad (40)$$

3. Если C - класс, то $$\llbracket t : C \rrbracket \overset{def}{=} \llbracket t \rrbracket \in C^* \quad (41)$$

4. $$\left\llbracket t : \bigcap_{i=1}^{m} T_i \right\rrbracket \overset{def}{=} \bigwedge_{i=1}^{m} \llbracket t : T_i \rrbracket \quad (42)$$

5. $$\llbracket t : T[x = y] \rrbracket \overset{def}{=} \llbracket t : T \rrbracket \quad (43)$$

1.2.1. СЕМАНТИКА АТРИБУТОВ КЛАССА

Если класс C является первичным, то в теорию F_M включается аксиома существования экземпляра:

$$\forall (x_1)...\forall (x_m)[R_C^*(x_1, ..., x_m) \rightarrow \rightarrow \exists (t)\mathrm{is}C(t, x_1, ..., x_m)], \quad (44)$$

где T_i – тип i-го параметра класса.

Если класс не является первичным, то утверждение включается в множество V как требующее верификации.

Если класс C отмечен атрибутом Unique, то в число аксиом включается аксиома единственности:

$$\forall (x_1)...\forall (x_m)\forall (t)\forall (t') [\mathrm{is}C(t, x_1, ..., x_m) \wedge \wedge \mathrm{is}C(t', x_1, ..., x_m) \rightarrow t = t']. \quad (45)$$

Если класс C неявно уникален, то утверждение включается в набор верификации V. В случае абстрактного класса утверждение не генерируется и не проверяется.

Для неабстрактного класса, кроме того, в теорию F_M добавляется функциональный символ $\mathrm{get}C(x_1, ..., x_m)$, описывающий экземпляр

класса с параметрами $x_1,...,x_m$ и характеризуемый аксиомой :

$$\forall(x_1)...\forall(x_m)\forall(t)[(\mathrm{get}\,C(x_1,...,x_m) = t) \leftrightarrow \mathrm{is}\,C(t,x_1,...,x_m)]. \quad (46)$$

Для замкнутого (в частности, дизъюнктного) класса генерируется аксиома , из которой следует, что объединение множеств экземпляров Case-**подклассов замкнутого класса совпадает** с множеством экземпляров самого класса C :

$$\forall(t)\left[t \in C^* \rightarrow \bigvee_{i=1}^{k} t \in S_i^*\right], \quad (47)$$

где $S = \{S_i\}_{i=1}^{k}$ - множество всех прямых подклассов C , отмеченных атрибутом Case.

Если класс C является дизъюнктным и $S = \{S_i\}_{i=1}^{k}$ - множество всех Case-**подклассов** C , то следующая совокупность утверждений (при всех $i = 1..k$) добавляется к числу аксиом теории:

$$\forall(t)\left[t \in S_i^* \rightarrow \neg \bigvee_{j=1, j\neq i}^{k} t \in S_j^*\right]. \quad (48)$$

Утверждения выражают тот факт, что никакой экземпляр дизъюнктного класса C не может одновременно быть экземпляром двух различных подклассов C .

1.2.2. ПРЕДИКАТЫ И ВЫЗОВЫ ОПЕРАЦИЙ

Для описания семантики операций потребуется вспомогательные понятия локального и полного предикатов, которые характеризуют ограничения, накладываемые на ее параметры (и результат, если речь идет о функции) в контексте ее описания и вызова соответственно.

Локальный предикат отношения определяется аналогично фактическому предусловию класса:

$$P_r(x_1,...,x_m) \overset{def}{=} \bigwedge_{i=1}^{m}\left([\![x_i : T_i]\!] \wedge [\![C_i(x_i)]\!]\right) \wedge \wedge [\![R(x_1,...,x_m)]\!], \quad (49),$$

где типы T_i, ограничения C_i и предусловие R относятся к параметрам отношения. В случае функции дополнительно учитывается ее постусловие и result-ограничение и тип результата, поэтому в общем случае локальный предикат функции зависит от result-переменной s :

$$P_r(x_1,...,x_m,s) \overset{def}{=} \bigwedge_{i=1}^{m}\left([\![x_i : T_i]\!] \wedge [\![C_i(x_i)]\!]\right) \wedge \wedge [\![R(x_1,...,x_m)]\!] \wedge [\![E(p_1,...,p_m,s)]\!] \wedge \wedge [\![s : T_0]\!] \wedge [\![C_0(s)]\!]. \quad (50)$$

Для обозначения *формального вызова операции* $r \in Op[T]$ *в контексте типа* T *и терма* t используется выражение $inv_C^r[t,x_1,...,x_k]$. Тип T может быть классом, производным типом или типом-пересечением. В случае класса C формальный вызов отношения (функции) непосредственно преобразуется в формулу (терм) с предикатным (функциональным) символом $\mathrm{rel}\,C.r$ (соответственно $\mathrm{fun}\,C.r$).

Полный предикат операции класса определяется следующим образом:

Если r - неявная операция, то ее полный предикат совпадает с полным предикатом в контексте прямого супертипа S :

$$P_r^T(x_1,...,x_m) \overset{def}{=} P_r^S(x_1,...,x_m). \quad (51)$$

Полный предикат собственной операции совпадает с локальным:

$$P_r^T(x_1,...,x_m) \overset{def}{=} P_r(x_1,...,x_m). \quad (52)$$

Полный предикат операции r , переопределяющей r' образуется путем соединения полного предиката r' с локальным предикатом r :

$$P_r^T(x_1,...,x_m) \overset{def}{=} P_{r'}^S(x_1,...,x_m) \wedge \wedge P_r(x_1,...,x_m). \quad (53)$$

Для функций определения , и формулируются аналогично.

Для операции производного типа $T = S[x = y]$ возможно два варианта:

1. Если операция r является неявной, то ее вызов в контексте T эквивалентен вызову в контексте S :

$$inv_T^r[t,x_1,...,x_k] = inv_S^r[t,x_1,...,x_k], \quad P_r^T(x_1,...,x_m) = P_r^S(x_1,...,x_m). \quad (54)$$

2. Вызов производной операции r эквивалентен вызову исходной операции r' в контексте типа S :

$$inv_T^r[t,x_1,...,x_k] = inv_S^{r'}[t,x_1,...,x_k], \quad P_r^T(x_1,...,x_m = P_{r'}^S(x_1,...,x_m). \quad (55)$$

Пусть теперь T - тип-пересечение $T = \bigcap_{i=1}^{m} T_i$ и $r \in Op[T]$. Если r - неявная операция типа T_{i_0} , то ее вызов в контексте типа T эквивалентен вызову в контексте типа T_{i_0} :

$$inv_T^r[t,x_1,...,x_k] = inv_{T_{i_0}}^r[t,x_1,...,x_k], \quad P_r^T(x_1,...,x_m) = P_r^{T_{i_0}}(x_1,...,x_m) \quad (56)$$

Полный предикат замкнутой объединенной операции представляет собой конъюнкцию

полных предикатов операций $r_i, i = 1..l$, входящих в порождающую ее группу родства R (операция r_i рассматривается в контексте компонента пересечения T^i):

$$P_r^T(x_1,...,x_m) \overset{def}{=} \bigwedge_{i=1}^{l} P_{r_i}^{T^i}(x_1,...,x_m), \qquad (57)$$

$$inv_T^r[t,x_1,...,x_k] = \\ = P_r^T(x_1,...,x_m) \wedge B(t,x_1,...,x_m). \qquad (58)$$

Для каждой абстрактной объединенной операции типа-пересечения создается специальный предикатный (либо функциональный) символ, обозначаемый $\mathrm{rel}\,T.r(t,x_1,...,x_k)$ (соответственно $\mathrm{fun}\,T.r(t,x_1,...,x_k)$). Вызов такой операции определяется аналогично вызову операции класса.

1.2.1. СЕМАНТИКА ОПЕРАЦИЙ КЛАССОВ И ТИПОВ-ПЕРЕСЕЧЕНИЙ

Операции, входящие в состав классов, непосредственно отображаются на предикаты и функции соответствующей формальной теории. Семантика операций, используемых в контексте других типов, определяется на уровне выражений вызова.

Пусть r - отношение с параметрами $P = \langle P_i \rangle_{i=1}^{k}$, входящее в состав класса C. Тогда в теорию модели включается $(k+1)$-арный предикат $\mathrm{rel}\,C.r(t,x_1,...x_k)$, свойства которого определяются в зависимости от вида отношения r:

1. Предикат неявного отношения эквивалентен формальному вызову соответствующей операции в контексте прямого супертипа S:

$$\forall(t)\forall(x_1)...\forall(x_k)[\mathrm{rel}\,C.r(t,x_1,...x_k) \leftrightarrow \\ \leftrightarrow inv_S^r[t,x_1,...,x_k]]. \qquad (59)$$

2. Собственное замкнутое отношение выражается непосредственно:

$$\forall(t)\forall(x_1)...\forall(x_k) \\ [\mathrm{rel}\,C.r(t,x_1,...x_k) \leftrightarrow \\ \leftrightarrow P_r(x_1,...,x_k) \wedge B(t,x_1,...,x_k)]. \qquad (60)$$

3. Для собственного абстрактного отношения формулируется только необходимое условие:

$$\forall(t)\forall(x_1)...\forall(x_k)[\mathrm{rel}\,C.r(t,x_1,...x_k) \rightarrow \\ \rightarrow P_r(x_1,...,x_k)]. \qquad (61)$$

4. Переопределяющее отношение описывается аксиомами

$$\forall(t)\forall(x_1)...\forall(x_k) \\ [\mathrm{rel}\,C.r(t,x_1,...x_k) \leftrightarrow \\ \leftrightarrow P_r^T(x_1,...,x_k) \wedge B(t,x_1,...,x_k)] \qquad (62)$$

и

$$\forall(t)\forall(x_1)...\forall(x_k) \\ [t \in C \rightarrow (\mathrm{rel}\,C.r(t,x_1,...x_k) \leftrightarrow \\ \leftrightarrow inv_S^{r'}[t,x_1,...,x_k])], \qquad (63)$$

где S - прямой супертип класса C, а r' - операция S, переопределенная операцией r.

Утверждение выражает полиморфное поведение операции (возможность вызова через экземпляр супертипа).

Если r - это функция, то для нее аналогичным образом создается функциональный символ $\mathrm{fun}\,C.r(t,x_1,...x_k)$, свойства которого определяются по аналогии с перечисленными выше свойствами отношений.

Предикатные и функциональные символы создаются также и для абстрактных объединенных операций типов-пересечений. Характеризующие их аксиомы имеют вид, аналогичный собственным абстрактным операциям классов.

ЗАКЛЮЧЕНИЕ

В рамках представленной работы рассмотривается оригинальный подход к решению проблемы разработки гибридного метаязыка JLS путем соединения логического и объектно-ориентированного подходов, приведено описание структуры типов и операций, введено определение семантики путем преобразования JLS-модели в формальную теорию над исчислением предикатов первого порядка с равенством.

СПИСОК ЛИТЕРАТУРЫ

1. *Jim Woodcock*. Using Z. Specification, Refinement and Proof. – Prentice Hall International Ltd., 1996. – 386 с.

2. *Hans-Erik Eriksson*. Business Modeling with UML. Business Patterns at Work. – John Wiley & Sons, 2000. – 459 с.

3. *Yingzhou Zhang*. A Survey of Semantic Description Frameworks for Programming Languages // – ACM SIGPLAN Notices. – 2004. – № 39.3 – С. 14 – 30.

4. *Лавров С.С.* Программирование. Математические основы, средства, теория. – СПб.: БХВ-Петербург, 2001. – 320 с.: ил.

5. *Чень Ч.* Математическая логика и автоматическое доказательство теорем: Пер. с англ. / Чень Ч, Ли Р. – М.: Наука. Главная редакция физико-математической литературы, 1983. – 360 с.

6. *Benjamin C. Pierce*. Types and Programming Languages. – The MIT Press, 2002. – 623 с.

7. *Стюарт Рассел*. Искусственный интеллект. Современный подход. 2-е издание: Пер. с англ. / Стюарт Рассел, Питер Норвиг – М.: Издательский дом "Вильямс", 2005. – 1424 с.

Тюкачев Николай Аркадиевич - канд. физ.-мат. наук, доц., зав. каф. "Программирование и информационные технологии" Воронежского государственного университета, Тел. (4732)208-470. E-mail: nik@cs.vsu.ru

Седунов Алексей Александрович – менеджер направления системных разработок ООО "Философт", тел. 8-961-182-31-28, e-mail: sedunov@km.ru.

Tukachev N.A. – candidat of phisics-math. Sciences, associate Proffessor, the dept. of the Programming and Information Technologies, Voronezh State University. Tel. (4732)208-470. E-mail: nik@cs.vsu.ru

Sedunov A.A. – manager of ООО **"Философт", Tel. 8-961-182-31-28, E-mail: sedunov@** km.ru.