УДК 004.4’24

О. А. Дерюгина

Московский государственный университет информационных технологий, радиотехники и электроники

Семантика и семантически эквивалентные трансформации UML-диаграмм классов

В статье формализуются такие понятия, как объектно-ориентированная архитектура ПС, диаграмма классов, класс, интерфейс, отношение наследования, агрегация и т.д. В статье описывается формальная семантика UML-диаграмм классов, на основе которой возможно выполнять сравнение двух диаграмм классов между собой, производить трансформации, проверяя инвариантность семантического значения.

Ключевые слова: архитектура программного обеспечения, проектирование программного обеспечения, объектно-ориентированная архитектура, UML, семантика UML-диаграмм, формальная семантика.

O. A. Deryugina

Moscow State Institute of Radio Engineering, Electronics and Automation

UML class diagrams: semantics and semantically equivivalent transformations

The paper formalizes such concepts as object-oriented software architecture, class diagram, class, interface, inheritance, aggregation, etc. The paper describes the formal semantics of UML class diagrams, which allows one to compare two class diagrams with each other, to check the semantic invariance after transformation.

Key words: software architecture, software architecture design, object-oriented architecture, UML, UML diagram semantics, formal semantics.

1. Введение

В связи со всё нарастающей сложностью современных программных систем (ПС) возникает потребность в формализации процесса проектирования архитектуры программного обеспечения. Подобная формализация позволит повысить степень автоматизации данного процесса.

Для проектирования архитектуры программного обеспечения часто используется унифицированный язык моделирования UML (Unified Modelling Language), позволяющий производить визуальное проектирование, документирование и формальное описание программных систем.

К преимуществам использования UML можно отнести наличие развитых средств проектирования UML-моделей (Enterprise Architect, Visual Paradigm и др.) и чётких общепринятых стандартов (стандарты группы OMG).

Язык UML позволяет проектировать архитектуру ПС, в том числе в рамках объектно-ориентированной парадигмы программирования. Модель ПС описывается в виде набора связанных между собой диаграмм (классов, прецедентов, компонентов, последовательности и т.д.) Каждая диаграмма состоит из элементов (классов, интерфейсов, пакетов, объектов и т.д.), связанных между собой отношениями (наследование, агрегация, зависимость и т.д.).

Для проведения исследований в области эволюционного проектирования объектно-ориентированной архитектуры ПС необходимо создание формальной теории, описывающей объектно-ориентированную архитектуру ПС и её семантику.

В статье описывается формальная семантика UML-диаграмм классов, на основе которой возможно выполнять сравнение двух диаграмм классов между собой, проверять инвариантность семантического значения после трансформаций в ходе эволюционного поиска (рис. 1).

На рис. 1 приведён пример семантически эквивалентной трансформации UML-диаграммы классов, заключающейся в введении для классов Designer (Дизайнер), Secretary (Секретарь) и Manager (Менеджер) с одинаковым набором полей: id (уникальный идентификатор), FIO (ФИО), address (адрес), positionID (идентификатор должности), salary (заработная плата) – вводится родительский класс Employee (сотрудник) с набором полей id, FIO, address, positionID, salary, от которого их наследуют классы Designer, Secretary и Manager.

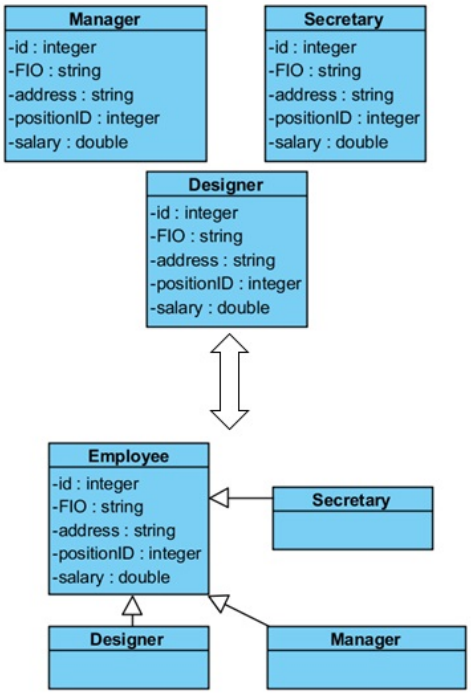

Рис. 1. Пример эквивалентной трансформации UML-диаграммы классов

2. Эволюционное проектирование объектно-ориентированной архитектуры

Объектно-ориентированное проектирование начинается с описания функциональности системы, а затем перечисления основных классов, методов и атрибутов, задания основных интерфейсов и иерархий наследования. Часто удобно применять готовые паттерны проектирования [1].

Различные подходы к объектно-ориентированному проектированию программного обеспечения методами поиска включают в себя: улучшение повторного использования существующих архитектур программного обеспечения через паттерны проектирования, построение иерархической декомпозиции для программной системы, проектирование структуры классов, проектирование архитектуры программного обеспечения на основе спецификаций.

В направлении создания инструментальных средств поддержки проектирования архитектуры программного обеспечения проводят исследования R. Outi [2], S. Mancoridis, B.S. Mitchell и др. [3], Di Penta [4] и др.

Важным этапом эволюционного поиска архитектуры программного обеспечения с улучшенными качествами является задание целевой функции. Часто используются комбиниро-

ванные целевые функции, в которых целевое значение складывается из нескольких факторов, каждому из которых задаётся свой вес. В ходе исследования веса могут настраиваться. Сами факторы (например, связность класса, зависимость классов друг от друга) рассчитываются на основе специальных метрик.

Обычно от пользователя требуется составить взвешенную целевую функцию, определяющую, какую архитектуру считать предпочтительнее.

Эволюционное проектирование архитектуры ПС относится к проектированию архитектуры на уровне PIM (Platform Independent Model – Платформо-независимая модель), поэтому характеристиками качества нельзя считать такие параметры, как оценка трудоёмкости выполнения операций, расходование памяти. Можно производить поиск архитектур с лучшими структурными качествами (баланс между максимальной внутренней связностью, минимальной внешней взаимозависимостью и требованием к модульному разбиению проекта).

Если говорить об оптимизации архитектуры систем на основе UML-диаграмм, то при трансформации UML-моделей в процессе поиска неизбежно возникает необходимость проверки корректности модели и семантической эквивалентности её оригиналу. В связи с этим встаёт вопрос о создании средств проверки UML-моделей на корректность и эквивалентность.

3. Теория объектно-ориентированной архитектуры программных систем

Для решения задачи сравнения двух диаграмм классов на предмет семантической эквивалентности автором статьи предложена семантика структуры UML-диаграмм классов. Для описания семантики UML-диаграмм классов формализуются такие понятия из парадигмы объектно-ориентированного проектирования, как архитектура ПС, диаграмма классов, класс, метод и др.

3.1. Понятие объектно-ориентированной архитектуры программной системы

Объектно-ориентированной архитектурой программной системы (ПС) называется архитектура A такая, что

$$A = \{M, RESTR, F, S, IN, OUT\},$$

где M – UML-модель ПС, $RESTR$ – множество ограничений к ПС, F – множество функциональных требований к ПС, S – семантическое значение ПС, IN – входные данные ПС, OUT – выходные данные ПС.

UML-моделью ПС называется формальная модель M такая, что

$$M = \{D_{classes}, D_{use\_cases}, D_{state\_charts}, D_{components}, D_{activity}, D_{objects}\},$$

где $D_{classes}$ – множество диаграмм классов, D_{use_cases} – множество диаграмм прецедентов, D_{state_charts} – множество диаграмм состояний, $D_{components}$ – множество диаграмм компонентов, $D_{activity}$ – множество диаграмм активностей, $D_{objects}$ – множество диаграмм объектов.

Множество диаграмм классов $D_{classes} = \{d_{class0}, ...d_{classm}\}$, где $d_{classi}, i \in 0, ..., m$ – диаграммы классов.

UML-диаграммой классов d_{class} называется такая UML-диаграмма, что

$$d_{class} = \{CL, INTR, REL\},$$

где CL – множество классов $CL = \{class_0, ...class_k\}$, $INTR$ – множество интерфейсов $INTR = \{interface_0...interface_l\}$, REL – множество связей $REL = \{relation_0...relation_g\}$.

Классом class называется такой элемент диаграммы классов $D_{classses}$, что

$$class = \{ATR, METH, P, ST, N\},$$

где ATR – множество атрибутов $ATR = \{atr_0...atr_n\}$, $METH$ – множество методов $METH = \{meth_0...meth_m\}$, P – класс-родитель (равен null, если родителей нет), ST – признак того, что класс статический (равен 0 или 1), N – идентификатор класса (имя).

Атрибутом называется такой элемент класса attribute, что

$$attribute = \{visibility, type, virtual, static, final, name\},$$

где $visibility = \{public, protected, private\}$ – видимость атрибута, $type$ – тип атрибута, $virtual$ – признак виртуальности, $static$ – признак статичности, $final$ – признак запрета изменения, $name$ – имя атрибута.

Методом называется такой элемент класса или интерфеса method, что

$$method = \{visibility, type, virtual, static, final, name, PAR, LOC\_PAR\},$$

где $visibility = \{public, protected, private\}$ – видимость метода, $type$ – тип возвращаемого значения, $virtual$ – признак виртуальности метода, $static$ – признак статичности метода, $final$ – запрет наследования, $name$ – имя метода, $PAR = \{parameter_0...parameter_n\}$ – множество входных параметров метода, $LOC\_PAR = \{parameter_0...parameter_m\}$ – множество локальных параметром метода, где $parameteri = \{type, name\}$, где $type$ – тип параметра, $name$ – имя параметра.

Интерфейсом $interface$ называется такой элемент UML-диаграммы классов, что

$$interface = \{METH, N\},$$

где $METH$ – множество методов $METH = \{meth_0...meth_m\}$, N – идентификатор интерфейса (имя).

Отношением $relation$ называется такая связь между элементами UML-диаграммы, что

$$relation = \{name, type, start, end, power\},$$

где $name$ – имя отношения, $type$ – тип отношения, $start$ – идентификатор начального участника отношения, end – идентификатор конечного участника отношения, $power$ – мощность отношения.

Множество диаграмм прецедентов $D_{use\_cases} = \{d_{use\_case_0}, ...d_{use\_case_m}\}$, где $d_{use\_cas_i}, i \in 0, ..., m$ – диаграммы прецедентов.

Диаграммой прецедентов $d_{use\_case_i}$ называется такая UML-диаграмма, что

$$d_{use\_case} = \{ACT, UC, REL\},$$

где ACT – множество акторов $ACT = \{actor_0, ...actor_k\}$, UC – множество прецедентов $UC = \{use\_case_0...use\_case_l\}$, REL – множество связей $REL = \{relation_0...relation_g\}$.

Актор $actor$, прецедент $usecase$ характеризуются именем. Отношение $relation = \{name, type, start, end, power\}$, где $name$ – имя отношения, $type$ – тип отношения, $start$ – начальный участник отношения, end – конечный участник отношения, $power$ – мощность отношения.

Множество диаграмм состояний $D_{state\_charts} = \{d_{state\_chart_0}, ...d_{state\_chart_m}\}$, где $d_{state\_chart_i}, i \in 0, ..., m$ – диаграммы состояний.

Диаграммой состояний d_{state_chart} называется такая UML-диаграмма, что

$$d_{state\_chart} = \{EV, ST, GC, ACT\},$$

где EV – множество событий $EV = \{event_0, ...event_k\}$, ST – множество состояний $ST = \{state_0...state_l\}$, GC – множество условий $GC = \{guard\_condition_0...guard\_condition_g\}$, ACT – множество действий $ACT = \{action_0...action_p\}$.

Множество диаграмм состояний $D_{components} = \{d_{components_0}, ...d_{components_m}\}$, где $d_{components_i}, i \in 0, ..., m$ – диаграммы компонетов.

Диаграммой компонетов $dcomponents$ называется такая UML-диаграмма, что

$$d_{components_i} = \{COMP, INTR, REL\},$$

где $COMP$ – множество компонетов $COMP = \{component_0, ...component_k\}$, $INTR$ – множество интерфейсов $INTR = \{interface_0...interface_l\}$, REL – множество связей $REL = \{relation_0...relation_g\}$.

Множество диаграмм активности $D_{activity} = \{d_{activity_0}, ...d_{activity_m}\}$, где $d_{activity_i}, i \in 0, ..., m$ – диаграммы активности.

Диаграммой активности называется такая UML-диаграмма $d_{activity}$, что

$$d_{activity} = \{ACT, DEC, SP, J\},$$

где ACT – множество действий $ACT = \{action_0, ...action_k\}$, DEC – множество решений $DEC = \{decision_0...decision_l\}$, SP – множество ветвлений конкурирующих деятельностей $SP = \{slpit_0...split_g\}$, J – множество ветвлений конкурирующих деятельностей $J = \{join_0...join_g\}$.

Множество диаграмм объектов $D_{objects} = \{d_{objects_0}, ...d_{objects_m}\}$, где $d_{objects_i}, i \in 0, ..., m$ – диаграммы объектов.

Диаграммой объектов $d_{objects}$ называется такая UML-диаграмма, что

$$d_{objects} = \{OBJ, INTR, REL\},$$

где OBJ – множество объектов $OBJ = \{object_0, ...object_k\}$, $INTR$ – множество интерфейсов $INTR = \{interface_0...interface_l\}$, REL – множество связей $REL = \{relation_0...relation_g\}$.

Множество ограничений к ПС $RESTR$ может быть представлено на одном из языков задания спецификаций к UML-модели ПС.

Множество функциональных требований к ПС $F = \{u_0...u_n\}$, где $u_i, i \in 0, ..., n$ – сценарии использования ($usecases$).

Семантическое значение ПС S может быть выражено в терминах какой-либо формальной семантики UML-диаграмм (аксиоматической, операционной и т.д.).

Входные данные IN – данные, подаваемые на вход ПС для последующей обработки.

Выходные данные OUT – данные, получаемые в результате работы ПС.

3.2. Семантика UML-диаграмм классов

Рассмотрим особенности семантики UML-диаграмм классов.

Пусть $C = \{\{attr_0, attr_1...attr_n\}, \{meth_0, meth_1...meth_m\}, static\}$ – класс с множеством атрибутов $attr_0, attr_1...attr_n$ и множеством методов $meth_0, meth_1...meth_m$, где static – признак статичности. $I = \{meth_0, meth_1, ...meth_m\}$ – интерфейс, реализующий множество методов $meth_0, meth_1, ...meth_m$.

Под конструктором класса C будем понимать метод $C()$, вызов которого $C.C()$ приводит к созданию в оперативной памяти объекта класса C.

Вызов конструктора класса возможен лишь у класса C, для которого $C.static = false$ (для нестатических классов).

Под деструктором класса C будем понимать метод $\sim C()$, вызов которого $C.\sim C()$ приводит к удалению из оперативной памяти объекта класса C.

Вызов деструктора класса возможен лишь у класса C, для которого $C.static = false$ (для нестатических классов).

Выражение $C1_{p1}\underrightarrow{assoc}C2_{p2}$ означает, что класс $C1$ связан отношением ассоциации мощностью $(p1,\ p2)$ (см. табл. 1) с классом $C2$, где $p1,\ p2$ – количество объектов класса $C1$ и $C2$ соответственно, участвующих в отношении.

Между классами возможны следующие виды отношений (*relations*): ассоциация (*association*), наследование – обобщение (*generalization*), агрегация (*aggregation*), композиция (*composition*), зависимость (*dependency*).

Между классом и интерфейсом возможно отношение реализации (*realization*).

Таблица 1

Мощность отношений между классами

Мощность отношения	($p1$, $p2$)
один к одному	(1,1)
один ко многим	$(1, 1 \dots n)$ или $(1, 0 \dots n)$
многие ко многим	$(1, 1 \dots n,\ 1, 1 \dots n)$ или $(0 \dots n,\ 0 \dots n)$
многие к одному	$(1, 1 \dots n,\ 1)$ или $(0 \dots n,\ 1)$

Например, запись $C1_1 \underrightarrow{assoc} C2_{0\dots n}$ означает, что каждый объект класса $C1$ связан отношением ассоциации с $0...n$ объектов класса $C2$.

Выражение $C1 \underrightarrow{gener} C2$ означает, что класс $C1$ наследует методы и атрибуты класса $C2$. Выражение $C1_{p1} \underrightarrow{aggreg} C2_{p2}$ означает, что класс $C1$ связан с классом $C2$ отношением агрегации.

Аксиома 1. *Пусть даны классы* $C1 = \{\{attr1_0, attr1_1 ... attr1_{n1}\}, \{meth1_0, meth1_1 ... meth1_{m1}\}\}$ *и* $C2 = \{\{attr2_0, attr2_1 ... attr2_{n2}\}, \{meth2_0, meth2_1 ... meth2_{m2}\}\}$.

Тогда

$S[\{C1 = \{\{attr1_0, attr11 ... attr1_{n1}\}, \{meth1_0, meth1_1 ... meth1_{m1}\}\}, C2 = \{\{attr2_0, attr2_1 ... attr2_{n2}\}, \{meth2_0, meth2_1 ... meth2_{m2}\}\}\}, \{...\}, \{C1 \underrightarrow{gener} C2, ...\}] \equiv$

$\equiv [\{C1 = \{\{attr1_0, attr1_1 ... attr1_{n1}\} \cup \{attr2_0, attr2_1 ... attr2_{n2}\}, \{meth1_0, meth1_1 ... meth1_{m1}\} \cup \{meth2_0, meth2_1 ... meth2_{m2}\}\}, C2 = \{\{attr2_0, attr2_1 ... attr2_{n2}\}, \{meth2_0, meth2_1 ... meth2_{m2}\}\}, \{\}, \{\}\}]$.

Аксиома 2. *Справедливо следующее выражение:*

$$C1_{p1} \underrightarrow{aggreg} C2_{p2} \to \exists atribute \in C1 : (type = C2) \vee (type = Container < C2 >),$$

где type – тип атрибута atribute класса $C1$, $Container < C2 >$ *– контейнер объектов класса* $C2$.

Выражение $C1_{p1} \underrightarrow{comp} C2_{p2}$ означает, что класс $C1$ связан с классом $C2$ отношением композиции.

Аксиома 3. *Справедливо следующее выражение:*

$$(C1_{p1} \underrightarrow{comp} C2_{p2}) \wedge (\sim C1()) \to \sim C2(),$$

где $\sim C1()$ *– вызов деструктора класса* $C1$, $\sim C2()$ *– вызов деструктора* $C2$.

Примечание. Аксиома 3 означает, что удаление из оперативной памяти объекта класса $C1$ автоматически приводит к удалению из памяти связанного с ним отношением композиции объекта класса $C2$. Выражение $C1 \underrightarrow{dep} C2$ означает, что класс $C1$ связан с классом $C2$ отношением зависимости.

Аксиома 4. *Справедливо следующее выражение:*

$$C1 \underrightarrow{dep} C2 \to (\exists method \in C1) \wedge ((\exists parameter \in PAR) \vee (\exists parameter \in LOC\_PAR)) : type = C2,$$

где PAR – множество формальных параметров метода method, LOC_PAR – множество локальных параметров метода method, type – тип параметра parameter.

Тогда будем считать, что структурная семантика UML-диаграммы классов описывается следующим образом:

$S = \{\{C1 = \{\{attr1_1 = \{\{attr1Type, isVirtual, isStatic, isFinal, attr1Name\}\}, attr1_2 = \{...\}, ...attr1_n = \{...\}\}, \{meth1_1 = \{\{type, isVirtual, isStatic, isFinal, name, PAR = \{parameter1_{1_1} = \{type, name\}, parameter1_{1_2} = = \{type, name\}...parameter1_{1_n} = \{type, name\}\}, LOC\_PAR = \{parameter1_{1_1} = \{type, name\}, parameter1_{1_2} = \{type, name\}...parameter1_{1_m} = \{type, name\}\}\}, \}, meth1_2 = \{...\}, ..., meth1_m = \{...\}\}, isStatic\}, C2 = \{...\}, ...Ck = \{\}\}, \{ I1 = \{\{meth1_1 = \{\{type, isVirtual, isStatic, isFinal, name, PAR = \{parameter1_{1_1} = \{type, name\}, parameter1_{1_2} = \{type, name\}... ...parameter1_{1_n} = \{type, name\}\}, LOC\_PAR = \{ parameter1_{1_1} = \{type, name\}, parameter1_{1_2} = \{type, name\}... ...parameter1_{1_m} = \{type, name\}\}\}, \}, meth1_2 = \{...\}, ..., meth1_m = \{...\}\}\}, I2 = \{...\}, ...Il = \{...\}\}, \{ R1, R2, ...Rk\} \}.$

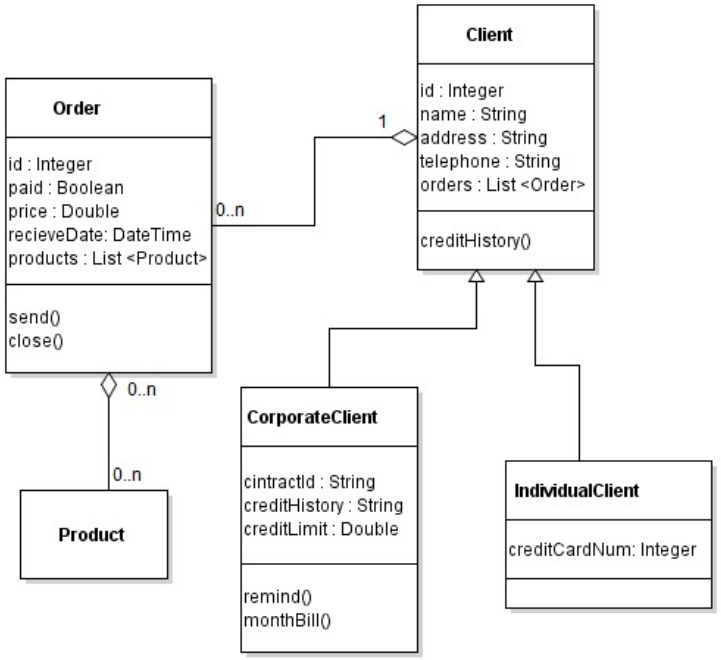

Рис. 2. Пример диаграммы классов

Например, семантическое значение диаграммы классов на рис. 2 будет следующим:
$S = \{\{C1 = \{\{atr1_1 = \{\{Integer, false, false, false, „id„\}\},$
$atr1_2 = \{\{String, false, false, false, „name„\}\}, atr1_3 = \{\{$
$String, false, false, false, „address„\}\},$
$atr1_4 = \{\{String, false, false, false, „telephone„\}\},$
$atr1_5 = \{\{List < Order >, false, false, false, „orders„\}\}\},$
$\{meth1_1 = \{void, false, false, false, „creditHistory„, PAR = \{\},$
$LOC\_PAR = \{\}\}, false, „Client„\},$
$C2 = \{\{\{atr1_1 = \{\{Integer, false, false, false, „id„\}\},$
$atr2_2 = \{\{Boolean, false, false, false, „paid„\}\},$
$atr2_3 = \{\{Double, false, false, false, „price„\}\},$
$atr2_4 = \{\{DateTime, false, false, false, „recieveDate„\}\},$
$atr2_5 = \{\{List < Product >, false, false, false, „products„\}\}\}, \{$
$meth2_1 = \{void, false, false, false, „send„, PAR = \{\}, LOC\_PAR = \{\}\},$
$meth2_2 = \{void, false, false, false, „close„, PAR = \{\},$
$LOC\_PAR = \{\}\}\}, false, „Order„\},$
$C3 = \{\{atr3_1 = \{\{String, false, false, false, „contractId„\}\},$

$atr3_2 = \{\{String, false, false, false, „creditHistory„\}\}$,
$atr3_3 = \{\{Double, false, false, false, „creditLimit„\}\}\}, \{$
$meth3_1 = \{void, false, false, false, „remind„, PAR = \{\}, LOC\_PAR = \{\}\}$,
$meth3_2 = \{void, false, false, false, „monthBill„, PAR = \{\}$,
$LOC\_PAR = \{\}\}\}, false, „CorporateClient„\}$,
$C4 = \{\{atr4_1 = \{\{Integer, false, false, false, „creditCardNum„\}\}\}$,
$\{\}, false, „IndividualClient„\}$,
$C5 = \{\{\}, \{\}, false, „Product„\}\}, \{\}, \{C1_1 \xrightarrow{aggreg} C2_{0..n}$,
$C2_{0..n} \xrightarrow{aggreg} C5_{0..n}, C3 \xrightarrow{gener} C1, C4 \xrightarrow{gener} C1\}\}$

Выражение $C1 \xrightarrow{real} I1$ означает, что класс $C1$ связан с интерфейсом $I1$ отношением реализации (реализует методы интерфейса $I1$), причём множество методов класса $C1$ M_{C1} содержит в себе множество методов интерфейса $I1$ M_{I1}:

$M_{I1} \subseteq M_{C1}$.

3.3. Семантически эквивалентные трансформации UML-диаграмм классов

Под трансформацией UML-диаграммы будем понимать такое изменение структурных элементов UML-диаграммы, которое обеспечивает инвариантность ее семантического значения S.

Теорема 1. (Следствие из Аксиомы 1) – Первая эквивалентная трансформация. *Пусть в UML-диаграмме классов dclasses дан класс-родитель* $C1 = \{\{attr1_1, attr1_2, ...attr1_n\}, \{meth1_1, meth1_2, ...meth1_m\}...\}$ *и даны классы* $C2 = \{\{attr2_i\}, \{meth2_j\}...\}...CN = \{\{attrN_i\}, \{methN_j\}...\}$. *Причем* $C2...CN \xrightarrow{gener} C1$. *Тогда*

$$S[d_{classes}] = S[\{C1 = \{\{attr1_1, attr1_2, ...attr1_n\}, \{meth1_1, meth1_2, ...meth1_m\}...\},$$
$$C2 = \{\{\{\{attr1_1, attr1_2, ...attr1_n\} \cup \{attr2_i\}\}, \{\{meth1_1, meth1_2, ...meth1_m\} \cup$$
$$\cup\{meth2j\}\}, ...\}, C3 = \{\{\{\{attr1_1, attr1_2, ...attr1_n\} \cup \{attr3_i\}\},$$
$$...\{\{meth1_1, meth1_2, ...meth1_m\} \cup \{meth3_j\}\}, ...\}, ...CN = \{\{\{\{attr1_1, attr1_2, ...$$
$$attr1_n\} \cup \{attrN_i\}\}, \{\{meth1_1, meth1_2, ...meth1_m\} \cup \{methN_j\}\}, ...\}\}, \{\}, \{\}\}],$$

где $attr2_i$, $attr3_i$, ... $attrN_i$ – *атрибуты классов C2 ... CN,* $meth2_j$, $meth3_j$, ... , $methN_j$ – *методы классов C2 ... CN.*

Доказательство.

Пусть $S[dclasses] = \{\{C1 = \{\{attr1_1, attr1_2, ...attr1_n\}, \{meth1_1, meth1_2, ...meth1_m\}...\}$, $C2 = \{\{attr2_i\}, \{meth2_j\}...\}, ...CN = \{\{attrN_i\}, \{methN_j\}\}, \{\}, \{C2 \xrightarrow{gener} C1$, $C3 \xrightarrow{gener} C1...CN \xrightarrow{gener} C1\}\}$.

Поочерёдно заменяя по Аксиоме 1 класс $Ci = \{\{attri_j\}, \{methi_k\}...\}$ и отношение наследования $Ci \xrightarrow{gener} C1$ в левой части на $Ci = \{\{attri_j\} \cup \{attr1_1, attr1_2, ...attr1_n\}, \{methi_k \cup \{meth1_1, meth1_2, ...meth1_m\}\}...\}$, получаем

$S[d_{classes}] = S[\{\{C1 = \{\{attr1_1, attr1_2, ...attr1_n\}, \{meth1_1, meth1_2, ...meth1_m\}...\},$
$C2 = \{\{attr1_1, attr1_2, ...attr1_n\} \cup \{attr2_i\}\}, \{\{meth1_1, meth1_2, ...meth1_m\} \cup \{meth2_j\}\}, ...\},$
$C3 = \{\{attr1_1, attr1_2, ...attr1_n\} \cup \{attr3_i\}\}, \{\{meth1_1, meth1_2, ...meth1_m\} \cup \{meth3_j\}\}, ...\}, ...$
$...CN = \{\{attr1_1, attr1_2, ...attr1_n\} \cup \{attrN_i\}\}, \{\{meth1_1, meth1_2, ...meth1_m\} \cup$
$\cup \{methN_j\}\}, ...\}\}, \{\}, \{\}\}]$.

Что и требовалось доказать.

Аксиома 5 (Вторая эквивалентная трансформация) – Интерфейс. *Справедливо следующее соотношение:*

$$S[\{\{C1, C2\}, \{I1\}, \{C2_{p1} \underrightarrow{dep} I1_{p2}, C1 \underrightarrow{real} I1\}\}] \equiv S[\{\{C1, C2\}, \{\emptyset\}, \{C2_{p1} \underrightarrow{dep} C1_{p2}\}\}].$$

Рассмотрим в качестве примера семантически эквивалентной трансформации UML-диаграммы классов шаблон проектирования Фасад, предложенный Гаммой [1].

Аксиома 6 (Третья эквивалентная трансформация) – Фасад (Facade). *Справедливо следующее соотношение:*

$$\begin{aligned} & S[\{\{C1, C2..CN\}, \{...\}, \{... \cup \{CK \underrightarrow{dep} \{C2 \vee C3 \vee .. \vee CN\} ... CM \\ & \qquad \underrightarrow{dep} \{C2 \vee C3 \vee .. \vee CN\}\}\}] \equiv \\ \equiv\ & S[\{\{C1 = \{\{\{attr1_i\} \cup \{C2..CN\}\}, \{\{meth1_i\} \cup \{\{meth2_j\} \cup \{meth3_k\} \cup \\ & \qquad \cup \{methN_l\}\}\}\}, \{...\}, \{... \cup \{CK ... CM \underrightarrow{dep} C1\}\}], \end{aligned}$$

где $C1$ *– класс-обёртка (фасад) для остальных классов* $CC2$ *...* CN*,* $attr1_i$ *– атрибуты класса* $C1$*,* $meth1_i$ *– методы класса* $C1$*,* $\{meth2_j\} \cup \{meth3_k\} \cup ... \cup \{methN_l\}$ *– методы,соответствующие вызову соответствующих методов, принадлежащих классам* $C2$ *...* CN *с параметром* $visibility = public$*,* CK *...* CM *– классы, связанные с классами* C2 *...* CN *отношением зависимости* (*dependency*).

4. Заключение

В работе рассматриваются вопросы эволюционного проектирования объектно-ориентированной архитектуры программного обеспечения.

Приведено формальное описание таких понятий из объектно-ориентированной парадигмы проектирования, как архитектура ПС, диаграмма классов, диаграмма прецедентов, класс, интерфейс, метод, отношение наследования и т.д.

Предложен формальный аппарат для описания семантики UML-диаграмм классов, позволяющий ввести понятие семантически эквивалентной трансформации диаграммы классов. Данный аппарат необходим для реализации эволюционного проектирования архитектуры программного обеспечения (в частности – поиска диаграммы классов с улучшенными структурными свойствами, семантически эквивалентной исходной).

Так же в работе на основе разработанного формального аппарата описываются семантически эквивалентные трансформации (наследование, введение интерфейса, введение предложенного Гаммой паттерна проектирования Фасад (Facade)).

По теме статьи автором будет сделан доклад на предстоящей II Международной научной конференции «Инжиниринг и телекоммуникации» Москва, 18–19 ноября 2015 года.

Литература

1. *Gamma E., Richard H., Ralph J. and John Vl.* Design patterns: Abstraction and reuse of object-oriented design // Springer Berlin Heidelberg, 1993.
2. *Raiha O.* Genetic Synthesis of Software Architecture University of Tampere Department of Computer Sciences // Lic. Phil. Thesis, September 2008.
3. *Mancoridis S., Mitchell B.S., Rorres C., Chen Y.F. and Gansner E.R.* Proc. International Workshop on Program Comprehension (IWPC 98). 1998. P. 45–53.
4. *Penta M. Di, Neteler M., Antoniol G. and Merlo E.* // The Journal of Systems and Software. V. 77, I. 3. 2005. P. 225–240.

References

1. *Gamma E., Richard H., Ralph J. and John Vl.* Design patterns: Abstraction and reuse of object-oriented design. Springer Berlin Heidelberg, 1993.
2. *Raiha O.* Genetic Synthesis of Software Architecture University of Tampere Department of Computer Sciences. Lic. Phil. Thesis, September 2008.
3. *Mancoridis S., Mitchell B.S., Rorres C., Chen Y.F. and Gansner E.R.* Proc. International Workshop on Program Comprehension (IWPC 98). 1998. P. 45–53.
4. *Penta M. Di, Neteler M., Antoniol G. and Merlo E.* The Journal of Systems and Software. V. 77, I. 3. 2005. P. 225–240.

Поступила в редакцию 01.06.2015.