

$$C = \{PvF \cup PvM, PtF \cup PtM, PbF \cup PbM\} \quad (1)$$

Г. Г. СЕРГЕЕВ, канд. техн. наук, доц. СевНТУ, г. Севастополь

ФОРМАЛИЗАЦИЯ ПРОЦЕССА РЕФАКТОРИНГА НА ОСНОВЕ СИМВОЛЬНОЙ ЗАПИСИ СТРУКТУРЫ КЛАССОВ

В статті пропонується символічний запис структури класів програмної системи як засобу абстрактного синтаксису та семантики понять об'єктного моделювання. Розглядаються операції над символічним записом і їх вживання в процесі оптимізації програмних систем.

В статье предлагается символная запись структуры классов программной системы как средства абстрактного синтаксиса и семантики понятий объектного моделирования. Рассматриваются операции над символной записью и их применение в процессе рефакторинга программных систем.

In this paper symbol record of class structure is offered. It is tools abstract syntax and semantics where objective design used. Operations above a symbol record are declaring. Process of code refactoring are supported by.

Введение. В настоящее время практически все программное обеспечение использует средства объектно-ориентированного программирования. Именно это объясняет исключительную важность задач, связанных с облегчением сопровождения и развития существующего программного кода [1]. В то же время, этим задачам уделяется недостаточное внимание, ощущается явный недостаток методик и эффективных инструментов поддержки работы с существующим кодом. При разработке объектно-ориентированных систем, необходимо решать две классических задачи: задачу анализа и задачу синтеза. Решение задач синтеза во многом автоматизировано средствами различных CASE систем, Framework и т.п. Для моделирования бизнес-процессов системного проектирования и отображения организационных структур в большинстве приложений используется язык UML. В работах [2, 3] рассматривается использование UML-диаграмм для решения задачи синтеза и декларативные подходы к решению задачи анализа. Однако решение задачи анализа на основе UML не формализовано, а сама диаграмма не дает возможности выполнять формальные преобразования над классами с целью рефакторинга проекта [4].

Формальное представление структуры классов программных объектов в виде символной записи. На основании материалов работы [2], можно утверждать, что класс – это триплет вида $C = \{Pv, Pt, Pb\}$, где Pv, Pt, Pb – множество частных, защищенных и публичных данных и методов класса. С другой стороны, основываясь на свойстве инкапсуляции, класс – это пара $C = \{F, M\}$, где F – множество полей, а M – множество методов класса. Таким образом,

где PvF, PtF, PbF – множество частных, защищенных и публичных полей класса, PvM, PtM, PbM – множество частных, защищенных и публичных методов класса. Очевидно, что $PvF \in F, PtF \in F, PbF \in F$, $PvM \in M, PtM \in M, PbM \in M$, $PvF \cap PtF = \emptyset$, $PvF \cap PbF = \emptyset$, $PtF \cap PbF = \emptyset$, $PvM \cap PtM = \emptyset$, $PvM \cap PbM = \emptyset$, $PtM \cap PbM = \emptyset$, $PvF \cup PtF \cup PbF = F$, $PvM \cup PtM \cup PbM = M$.

Для описания структуры класса в виде символической записи предлагается следующая нотация. Символ, обозначающий атрибут класса, записывается как N^{type} , где N – имя поля, а $type$ – тип поля. Метод класса предлагается записывать как $Meth_{prim}^{type}(param)$, где $Meth$ – идентификатор метода, $param$ – перечень формальных параметров, а $prim \subseteq [virtual, abstract, override]$. Допустимо, что $prim = \emptyset$ и/или $param = \emptyset$. Класс определяется как $Cname_{Cprim}$, где $Cprim = abstract | \emptyset$. Синтаксис символной записи структуры класса определяется грамматикой, представленной в таблице 1.

Таблица 1 – Грамматика символной записи структуры класса

$\langle ClassDeclare \rangle \rightarrow \langle CN \rangle = \langle CNbase \rangle + \{ \{ \langle PrvF \rangle, \langle PrvM \rangle \}; \{ \langle PrtF \rangle, \langle PrtM \rangle \}; \{ \langle PubF \rangle, \langle PubM \rangle \} \}$	
$\langle CN \rangle \rightarrow \langle id \rangle \langle Cprim \rangle$	$\langle FIELD \rangle \rightarrow \langle id \rangle \langle type \rangle$
$\langle PrvF \rangle \rightarrow \langle FIELDS \rangle$	$\langle FIELD \rangle \rightarrow \epsilon$
$\langle PrtF \rangle \rightarrow \langle FIELDS \rangle$	$\langle METHODS \rangle \rightarrow \langle METHOD \rangle$
$\langle PubF \rangle \rightarrow \langle FIELDS \rangle$	$\langle METHODS \rangle \rightarrow \langle METHOD \rangle,$
$\langle FIELDS \rangle \rightarrow \langle FIELD \rangle$	$\langle METHODS \rangle$
$\langle PrvM \rangle \rightarrow \langle METHODS \rangle$	$\langle METHOD \rangle \rightarrow \langle id \rangle \langle type \rangle \langle prim \rangle \langle param \rangle ()$
$\langle PrtM \rangle \rightarrow \langle METHODS \rangle$	$\langle Cprim \rangle \rightarrow abstract \epsilon$
$\langle PubM \rangle \rightarrow \langle METHODS \rangle$	$\langle prim \rangle \rightarrow virtual abstract override \epsilon$
$\langle FIELDS \rangle \rightarrow \langle FIELD \rangle, \langle FIELDS \rangle$	$\langle CNbase \rangle \rightarrow \langle id \rangle + \epsilon$

Таким образом, в символном виде описание нового класса $newClass$, порожденного от базового класса $baseClass$ это выражение вида

$$newClass = baseClass + \{ \{ PvF, PvM \}, \{ PtF, PtM \}, \{ PbF, PbM \} \}$$

Рассмотрим пример символной записи структуры классов, показанной в виде UML-диаграммы на рис. 1.

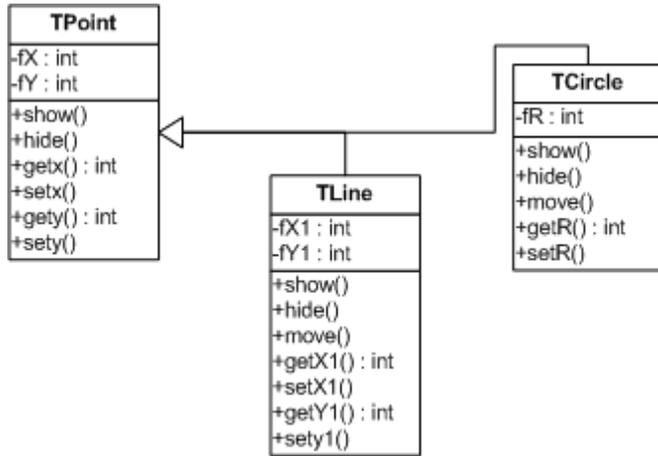


Рис. 1 – UML-диаграмма представления классов геометрических фигур

$$TPoint = \left\{ \begin{array}{l} [fX^{int}, fY^{int}], [], [show_{virtual}(), hide_{virtual}(), getX_{virtual}(), \\ setX_{virtual}(newX^{int}), getY_{virtual}(), setY_{virtual}(newY^{int})] \end{array} \right\},$$

$$TLine = TPoint + \left\{ \begin{array}{l} [fX1^{int}, fY1^{int}], [], [show_{override}(), hide_{override}(), \\ getX1_{virtual}(), setX1_{virtual}(newX1^{int}), getY1_{virtual}(), \\ setY1_{virtual}(newY1^{int})] \end{array} \right\},$$

$$TCircle = TPoint + \left\{ \begin{array}{l} [fR^{int}], [], [show_{override}(), hide_{override}(), getR_{virtual}(), \\ setR_{virtual}(newR^{int})] \end{array} \right\}.$$

Очевидно, что символьная запись структуры классов компактнее соответствующей ей UML-диаграммы. Определим на основании предложенной символьной записи операции над классами для решения задачи анализа структуры проекта и его последующего рефакторинга.

Определим для двух произвольных классов $C1$ и $C2$ операцию пересечения, которую будем записывать как $C1 \cap C2$. Введем в рассмотрение функцию $I(\chi_1, \chi_2) = [\chi_1 \setminus (\chi_1 \cap \chi_2), \chi_2 \setminus (\chi_1 \cap \chi_2)]$. Пусть операция $C1 \cap C2$. в соответствии со спецификацией (1) определяется как:

$$baseC = \left\{ \begin{array}{l} [PvF_{C1} \cap PvF_{C2}, PvM_{C1} \cap PvM_{C2}] \\ [PtF_{C1} \cap PtF_{C2}, PtM_{C1} \cap PtM_{C2}] \\ [PbF_{C1} \cap PbF_{C2}, PbM_{C1} \cap PbM_{C2}] \end{array} \right\},$$

$$C1 \cap C2 = baseC + \left\{ \begin{array}{l} [I(PvF_{C1}, PvF_{C2}), I(PvM_{C1}, PvM_{C2})] \\ [I(PtF_{C1}, PtF_{C2}), I(PtM_{C1}, PtM_{C2})] \\ [I(PbF_{C1}, PbF_{C2}), I(PbM_{C1}, PbM_{C2})] \end{array} \right\}.$$

Базовый класс $baseC$ – это ближайший общий предок в дереве иерархии или единый базовый класс $BaseObject$ ($NSObject$, $java.lang.Object$, $System.Object$, $TObject$ и т. п.), если такого не существует. Если реализация методов с одинаковым объявлением PvM, PtM, PbM различна для операндов операции пересечения, то в результате пересечения они получают дополнительный префикс $abstract$. Проиллюстрируем операцию строгого пересечения на примере UML-диаграммы классов, представленных на рис. 2.

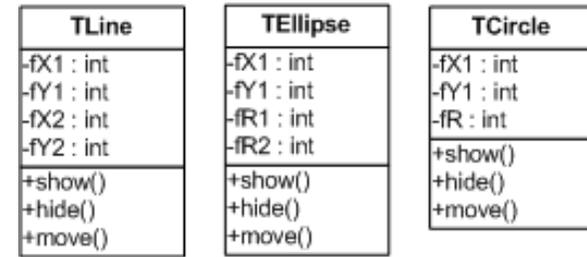


Рис. 2 – UML-диаграмма классов без общего предка

$$TEllipse \cap TLine = BaseObject + \left\{ \begin{array}{l} [fX1^{int}, fY1^{int}] [] \\ [show_{virtual, abstract}(), \\ hide_{virtual, abstract}(), move()] \end{array} \right\},$$

$$TEllipse \cap TCircle = TEllipse \cap TLine,$$

$$TCircle \cap TCircle = TEllipse \cap TLine.$$

Результатом пересечения классов, как правило, является новый класс или общий предок классов, заявленных в качестве операндов. Таким же образом определяем операцию вычитания классов $C1 \setminus C2$:

$$C1 \setminus C2 = \left\{ \begin{array}{l} [PvF_{C1} \setminus PvF_{C2}, PvM_{C1} \setminus PvM_{C2} + \forall_{Mi \in C1} Mi_{abstract}] \\ [PtF_{C1} \setminus PtF_{C2}, PtM_{C1} \setminus PtM_{C2} + \forall_{Mi \in C1} Mi_{abstract}] \\ [PbF_{C1} \setminus PbF_{C2}, PbM_{C1} \setminus PbM_{C2} + \forall_{Mi \in C1} Mi_{abstract}] \end{array} \right\}.$$

То есть операция вычитания есть традиционное вычитание множеств полей и методов класса с учетом необходимости перегрузки абстрактных методов.

Решение задачи рефакторинга на основе символьной записи.

Образование новых классов в результате выполнения операции пересечения является основой для процесса рефакторинга, описанного в [3]. При этом следует действовать по следующему формальному алгоритму вне зависимости от лингвистических сред и особенностей реализации.

Шаг 1. Выполнить преобразование в символьную запись всех объявлений классов программной системы.

Шаг 2. Для всех классов, имеющих общего предка в дереве иерархии выполняется операция строгого пересечения: $TParentObject = \bigcap^i C_i$.

Шаг 3. Если результат операции пересечения не равен $TParentObject \neq BaseObject$, определяем новый класс предка.

Шаг 4. Выполняем рефакторинг кода путем вычитания из каждого класса C_i класса $TParentObject$.

Рассмотрим пример рефакторинга программных классов, представленных на рисунке 2.

$$TFigure = BaseObject + \left\{ \left[\begin{array}{l} [fX1^{int}, fY1^{int}] [] \\ [show_{virtual,abstract}(), hide_{virtual,abstract}(), move()] \end{array} \right] \right\},$$

$$TLine = TFigure + \{ [fX2^{int}, fY2^{int}], [], [show_{override}(), hide_{override}()] \},$$

$$TEllipse = TFigure + \{ [fR1^{int}, fR2^{int}], [], [show_{override}(), hide_{override}()] \},$$

$$TCircle = TFigure + \{ [fR^{int}], [], [show_{override}(), hide_{override}()] \}.$$

Выводы. Формальный подход к решению задачи рефакторинга программных систем на основе символьной записи структуры классов является основой для построения программного продукта, позволяющего выполнять автоматизированный рефакторинг архитектуры программных систем практически без участия программиста.

Список литературы: 1. Ксензов М. В. Рефакторинг архитектуры программного обеспечения / М. В. Ксензов // Труды института системного программирования РАН. – 2004. – Т. 8. – С. 180–192. 2. Буч Г. Язык UML. Руководство пользователя / Д. Рамбо, А. Джексонсон. – СПб. : ДМК Пресс, 2007. – 435 с. 3. Буч Г. Объектно-ориентированный анализ и проектирование приложений на C++ / Г. Буч. – М. : Бинум, 1998. – 560 с. 4. Фаулер М. Рефакторинг: улучшение существующего кода / М. Фаулер. – СПб. : Символ, 2003. – 432 с.