УДК 51.681.3

Национальный авиационный университет
Н.А.Сидоров, М.Г.Луцкий, И.В.Гученко

Системная инженерия программного обеспечения

В статье предлагается метод решения задачи системного описания программного обеспечения на основе уровней инкапсуляции и алгебры систем. Предложена методика формального описания программных капсул как систем. Показана возможность формализации мегамодульного уровня инкапсуляции, рассматривая программную систему как систему систем.

У статті пропонується метод рішення задачі системного опису програмного забезпечення на основі рівнів інкапсуляції та алгебри систем. Запропонована методика формального опису програмних капсул як систем. Показана можливість формалізації мегамодульного рівня інкапсуляції, розглядаючи програмну систему як систему систем.

In the article the method deciding task of the software system description beth on the basis of encapsulation levels and the systems of algebras. Methodology of formal specification of capsules as systems is offered. Possibility of formalization of megamodule's encapsulation level is showed. Examining the program system as system of systems is presented.

Ключевые слова: инженерия программного обеспечения, система, уровень инкапсуляции, капсула, алгебра процессов реального времени, система систем

Введение

В инженерии программного обеспечения важно уметь конструктивно представлять программы [1]. Используя алгебру процессов реального времени можно формально описать поведение программной системы конечным множеством метаповедений. Метаповедения моделируются с помощью 17 метапроцессов и их композиций, а также алгебраических отношений [2]. Метод, предложенный для решения задачи представления программы как композиции систем-капсул, позволяет формально описать любое программное обеспечение в терминах открытых систем и исследовать его свойства, например, критическую массу системы, выполняемую работу, мощность, эффективность, равновесие системы, диссимиляцию.

1. Представление программы как композиции систем-капсул

Рассматривается задача – используя конструктивное представление структуры программы как шести уровней инкапсуляции [1] (рис. а) и алгебру систем [2], описать программу как композицию систем-капсул. При этом каждая капсула рассматривается как открытая система и описывается с помощью теории множеств, теории отношений, логики высказываний и алгебры процессов реального времени. Большинство практических систем реального мира являются открытыми. Они должны взаимодействовать с внешним миром (среда Θ) для обмена энергией, веществом и/или информацией. Типичными проводниками взаимодействия открытой системы со средой являются вход и выход.

Будем описывать капсулу как кортеж следующего вида:

$$S = (C, R, B, \Omega, \Theta) = (C, R^c, R^i, R^o, B, \Omega, \Theta),$$

где Θ – внешняя среда с непустым множеством компонентов $C_\theta, C_\theta \cap C = \emptyset$;

$R^c \subseteq C \times C$- множество внутренних отношений;

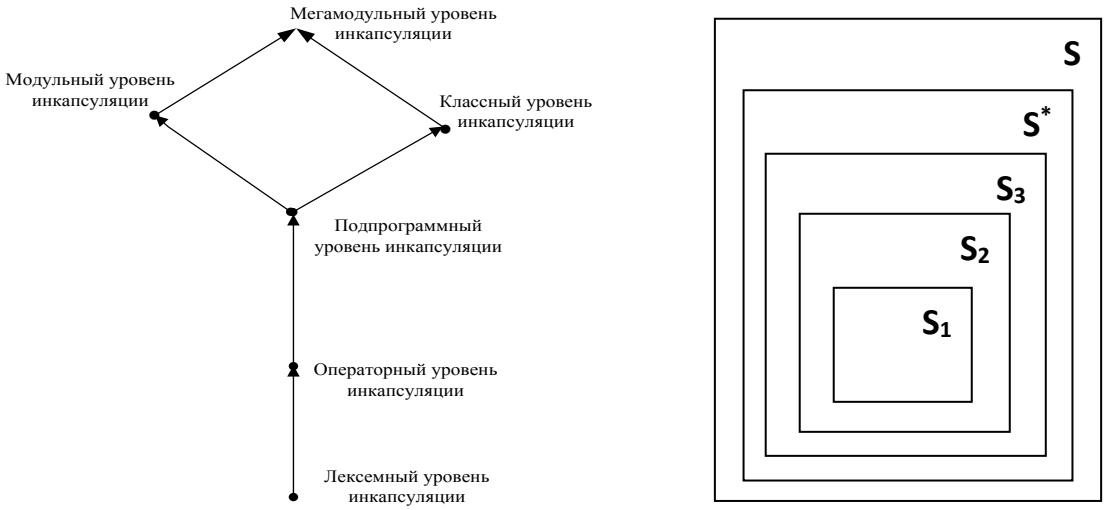
$R^i \subseteq C_\theta \times C$- множество внешних входных отношений;

$R^o \subseteq C \times C_\theta$- множество внутренних исходящих отношений.

B – множество поведений (или функций) $B = \{b_1, b_2, \dots, b_p\}$;

Ω – множество структур на компонентах, условия отношений, рамки (область действия) поведения, $\Omega = \{\omega_1, \omega_2, \dots, \omega_q\}$.

Композиция систем является наиболее сложной операцией и возможна лишь между открытыми системами [2]. Она интегрирует две или больше систем в суперсистему с иерархической архитектурой.



а) б)

Рис. а) Взаимосвязь уровней инкапсуляции
б) Схема композиции систем-капсул

Используя алгебру систем, можно представить программу как композицию капсул (рис.б) следующим образом:

$S=S^{*}>\!\!\rightarrow S_3=(S_{31}||S_{32})>\!\!\rightarrow S_2=(S_{20}||S_{21}||S_{22}||S_{23})>\!\!\rightarrow >\!\!\rightarrow S_1$,

где

– || - отношение параллельности, которое используется для описания композиции между функционально параллельными или эквивалентными системами;

– $>\!\!\rightarrow$ - отношение вложенности, которое определяет одну систему как встроенную часть другой;

– $S^{*}= S_4 \vee S_5$, где:

– S_5 – система "Класс", описывающая капсулу классного уровня инкапсуляции;

– S_4 – система "Модуль", описывающая капсулу модульного уровня инкапсуляции.

– S_3 – система "Подпрограмма", которая представлена двумя системами S_{31} и S_{32} (открытой и закрытой подпрограммами соответственно), описывает капсулу подпрограммного уровня инкапсуляции;

– S_2 – система «Оператор», описывающая капсулу операторного уровня инкапсуляции. Состоит из четырех подсистем типа "Оператор", а именно:

– S_{20} - оператор присваивания;

– S_{21}, S_{22} и S_{23} - операторный базис (операторы последовательности, выбора и повторения соответственно).

– S_1 – система "Лексема", описывающая капсулу лексического уровня инкапсуляции.

2. Методика формального описания капсул как систем

Будем формально описывать каждую капсулу как систему по следующей схеме, состоящей из шести действий.

1). Представить систему в виде кортежа:

$$S = (C, R, B, \Omega, \theta) = (C, R^0, R^0, R^0, B, \Omega, \theta),$$

где C – непустое множество компонентов системы, $C = \{c_1, c_2 ..., c_n\}$;

$R^0 \subseteq C \times C'$ - множество внутренних отношений;

$R^0 \subseteq C_\theta \times C$ - множество внешних входных отношений;

$R^0 \subseteq C \times C_\theta$ - множество внутренних исходящих отношений;

B – множество поведений (или функций) $B = \{b_1, b_2, ..., b_n\}$;

Ω - множество структур на компонентах, условия отношений, рамки (область действия) поведения, $\Omega = \{\omega_1, \omega_2, ..., \omega_n\}$;

θ - внешняя среда с непустым множеством компонентов $C_\theta, C_\theta \cap C = \varnothing$;

2). Формально описать каждый элемент кортежа, применяя:

– для C, θ – теорию множеств и алгебраические операции над ними;

– для R – теорию отношений, обязательно проверяя выполнение свойств асимметричности и рефлексивности, так как в случае невыполнения хотя бы одного из свойств кортеж нельзя считать системой;

– для B, Ω – логику высказываний и алгебру процессов реального времени.

3). Учесть то, что поведение программной системы является функцией от вычислительных операций, времени и структур памяти.

В алгебре процессов реального времени программные системы рассматриваются как обладающие статическим и динамическим поведением. Статическое поведение фиксируется во время работы компилятора и моделируется с помощью множества процессов и их отношений. Каждый процесс представляется путем описания функциональности, входных и выходных данных, а также взаимосвязи с другими процессами. Динамическое поведение определяется во время выполнения программы. При этом различаются процессы базового и высокого уровней [2]. Последние имеют место в системах реального времени.

4). Выбрать и формально описать свойства системы, например, размер, амплитуда, сложность, связность, сцепление:

Размер S_s, определяется мощностью множества C

$$S_s = \#C = n_c,$$

то есть размер системы равен числу компонентов системы;

Амплитуда M_s –

$$M_s = \#R = n_r = \#(C \times C) = n_c^2,$$

то есть равна числу асимметричных бинарных отношений между n_c компонентами системы, включая рефлексивные отношения;

Сложность –

$$C_{\max} = 2^{M_S}$$

$$C_r = M_s' = n_c(n_c - 1),$$

где M_s' - число бинарных отношений системы, не включая рефлексивные;

Связность (только для открытых систем)

$$CH(S) = \frac{\#R^c(S)}{\#R^c(s) + \#R^i(s) + \#R^0(s)} \cdot 100\%;$$

Сцепление (только для открытых систем)

$$CP(S) = \frac{\#R^i(s) + \#R^0(s)}{\#R^c(s) + \#R^i(s) + \#R^0(s)} \cdot 100\%$$

$$CH(S) + CP(S) = 100\%.$$

5). Классифицировать систему, исходя из оценки свойств (п. 5) и ключевых характеристик элементов кортежа [2].

6). Каждую капсулу, которая имеет подсистемы, рассмотреть как суперсистему и описать формально с помощью операций алгебры систем.

3. Пример 1. Применение метода для описания капсул

Приведем пример представления капсул операторного уровня инкапсуляции как систем [1]. Использованные для представления капсул обозначения представлены в приложении 1.

Капсула операторного уровня инкапсуляции описывается следующим образом:

$$S_2 \equiv Operator \hat{=} (S_{20} \| S_{21} \| S_{22} \| S_{23})$$

Система S_2 – «Оператор» представлена четырьмя системами типа «Оператор»:

$S_{20} \equiv$ Оператор присваивания,

$S_{21} \equiv$ Оператор последовательности –

$S_{22} \equiv$ Оператор выбора

$S_{23} \equiv$ Оператор повторения

$S_{21} - S_{23}$ операторный базис язика программирования

3.1 Оператор присваивания S_{20}

$S_{20} = \left\{C_{20}, R_{20}^c, R_{20}^i, R_{20}^0, B_{20}, \Omega_{20}, \theta_{20}\right\}$ - открытая система.

$C_{20} = \{l_j \mid l_j = l_j(\alpha_i) = \alpha_{j1}...\alpha_{jn},\ i = \overline{1,n},$

$j = \overline{1,m},\ \ m,n \in N,\ \alpha_i \in A\}$, где

l_j – лексема, α_i – символы алфавита A языка программирования.

$R_{20}^c = \{\frown\}$, где $\frown$ - отношение последовательного объединения.

$(\forall l_j \in \alpha, j \in N)\ \ (l_1 \frown l_2 \frown ... \frown l_m = l_1 l_2 ... l_m, m \in N),$

α - множество лексем.

Отношение $\frown$ рефлексивное, асимметричное, антитранзитивное.

R_{20}^i = {описать оператор (программист; лексема)}

R_{20}^0 = {изменить вектор состояния (лексема; транслятор)}

θ_{20} = {программист, транслятор}

B_{20} = {изменить значение переменной}

S_{20}.StaticBehaviors $\hat{=}$ SysInitial

$\rightarrow$ GetAddress (< I:: (l_j) >; < O:: (z_j^n, Ⓢ AddressGotten**BL**) >)

>> Dereference (< I:: (z_j^n) >; <O:: (z_j^c, Ⓢ VarContMeanAcess**BL**) >)

$\rightarrow$ GetMeanR (< I:: (l_j) >; < O:: (z_j, Ⓢ MeanRAccess**BL**) >)

>> ReplaceMean (< I:: (z_j)>;

<O:: ($\overline{z_j^c}$; Ⓢ VarMeanReplaced**BL**; Ⓢ VectorChanged**BL**)>).

Развернутое описание S_{20}.StaticBehaviors:

// State 1 (PN**N**:=1)

GetAddress (<I:: (l_j) >; <O:: (z_j^n, Ⓢ AddressGotten**BL**) >)

{(♦ ProgRunning**BL**=T

$\rightarrow$ NameOp**BL**=T // базовая операция “имя”

$\rightarrow$ MEM [ptr P]**BL** -> VarNameMean**BL**

$\rightarrow$ Ⓢ AddressGotten**BL** = T // взятие адреса выполнено

| ♦~

$\rightarrow$ Ⓢ AddressGotten**BL** = F)}

// State 2 (PN**N**: = 2)

Dereference (<I:: (z^n_j) >; < O:: (z^c_j , Ⓢ VarContMeanAccess**BL**)>)

{(♦ ProgRunning**BL** = T $\wedge$ Ⓢ AddressGotten**BL** = T

$\rightarrow$ DenameOp**BL** = T // базовая операция “разыменование”

$\rightarrow$ Ⓢ VarContMeanAccess**BL** = T // значение содержания переменной получено

| ♦ ~

$\rightarrow$ Ⓢ VarContMeanAccess**BL** = F)}

// State 3 (PN**N**: = 3)

GetMeanR (<I:: (l_j) >; < O:: (z_j), Ⓢ MeanRAccess**BL**) >)

{(♦ ProgRunning**BL** = T $\wedge$ Ⓢ VarContMeanAccess**BL** = T

$\rightarrow$ (NameOp**BL** = T $\wedge$ DenameOp**BL** = T) $\vee$ LiteralOp**BL** = T

$\rightarrow$ Ⓢ MeanRAccess**BL** = T // получен доступ к значению правой части оператора

| ♦ ~

$\rightarrow$ ⓈMeanRAccess**BL** = F)}

// State 4 (PN**N**: = 4)

ReplaceMean (<I:: (z_i) >; <O:: ($\overline{z_j^c}$,Ⓢ VarMeanReplace**BL**, VectorChanged**BL**) >)

{(♦ ProgRunning**BL** = T $\wedge$ ⓈMeanRAcess**BL** = T $\wedge$ MeansTypeEq**BL** = T

$\rightarrow$ Ⓢ VarMeanReplaced**BL** = T

$\rightarrow$ Ⓢ VectorChanged**BL** = T

| ♦~

$\rightarrow$ Ⓢ VarMeanReplaced**BL** = F $\wedge$ VectorChanged**BL** = F)}

S_{20}.Dynamic Behaviors $\hat{=}$ {§$\rightarrow$

(@‘ PN**N** = 1’ $\hookrightarrow_e$ GetAddress (<I:: (l_j) > ; <O:: (z_j , Ⓢ AddressGotten**BL**) >)

| @ ‘PN**N** = 2’$\hookrightarrow_e$ Dereference (<I:: (z_j^n) >; <O:: (z_j^c, Ⓢ VarContMeanAccess**BL**) >)

| @ ‘PN**N** = 3’$\hookrightarrow_e$ GetMeanR (<I:: (l_j) >; <O:: (z_j, Ⓢ MeanRAccess**BL**) >)

| @ ‘PN**N** = 4’ $\hookrightarrow_e$ ReplaceMean (<I:: (z_j) >; <O:: $\overline{z_j^c}$, ⓢVarMeanReplaced**BL**, VectorChanged**BL**) >)
$\rightarrow$§}

$\Omega = \{\psi\}$,

где ψ - преобразование “действие оператора присваивания”

$\overline{X_0} \xrightarrow{\psi} \overline{X'}$, то есть выполнение оператора присваивания изменяет вектор состояния программы.

$\overline{X} = (x_1, x_2, ..., x_p)$, $p \in N$, x_i - фиксированные точки, соответствующие завершению операторов программы.

VectorChanged**BL** = T $\equiv x_i^0(a_i^0; z_j^c; t_i^0) \xrightarrow{\psi} x_i'(a_i'; z_j; t_i') \Rightarrow \overline{x_0} \xrightarrow{\psi} \overline{X'}$,

где a_i - имя программного объекта, z_j^c, z_j - состояние значения содержания программного объекта, t_i - момент времени.

Абстрактная, конечная (#C $\neq \infty$), открытая, дискретная, динамическая система.

а) Размер S_S= #C = $n_C \in [1,10) \cup [10,100)$ - малая или средняя система;

б) Амплитуда

$$M_s = n_c^2 \in [1,10^2) \cup [10^2, 10^4);$$

в) Сложность

$C_{\max} = 2^{M_s} \in [2, 2^{100}] \cup (2^{100}, 2^{10,000})$,

$C_r = M_S' = n_C(n_C - 1) \in [0, 90] \cup (90, 0{,}99 \cdot 10^4]$;

г) Связанность

$$CH(S_1) = \frac{\#R^c}{\#R^c + \#R^i + \#R^0} \cdot 100\% = \frac{1}{3} \times 100\% \approx 33\%;$$

д) Сцепление

$$CP(S_1) = \frac{\#R^i + \#R^0}{\#R^c + \#R^i + \#R^0} \cdot 100\% = \frac{2}{3} \times 100\% \approx 67\%,$$

$$CH(S_1) + CP(S_1) = 33\% + 67\% = 100\%$$

3.2 Оператор последовательного выполнения S_{21}

$S_{21} \hat{=} (C_{21}, R_{21}^c, R_{21}^i, R_{21}^o, B_{21}, \Omega_{21}, \theta_{21})$.

$C_{21} = \{l_j \mid l_j = l_j(\alpha_i) = \alpha_{j1} ... \alpha_{jn},\ i = \overline{1,n},\ \ j = \overline{1,m},\ \ n,m \in N,\ l_j \in \alpha,\ \alpha_i \in A\}$, где

l - множество лексем l_j,

A - алфавит языка программирования,

α_i - символы алфавита.

$R_{21}^c = \{\frown\}$ - отношение последовательного соединения: рефлексивное, асимметричное, антитранзитивное.

$R_{21}^i = \{$ описать оператор (программист; лексема)$\}$

$R_{21}^0 = \{$ изменить вектор состояния (лексемы; транслятор)$\}$

$\theta_{21} = \{$ программист, транслятор$\}$

$B_{21} = \{$ образовывать последовательность$\}$

S_{21}. StaticBehaviors $\hat{=}$ Sysinitial
$\rightarrow$ Sequental Execution

Sequental Execution (<I:: (l_j) >; <O:: (ⓢSequenceCreated**BL**, VectorChanged**BL**)>)

{ $\underset{P_{k+1}ExistBL=T}{\overset{F}{R}}$ $P_k >> P_{k+1}$ // P_k, P_{k+1} - операторы преобразования

$\rightarrow$ ⓢSequenceCreated**BL**=T

⟶ VectorChanged**BL**=T}

S_{21}. DynamicBehaviors ≙ {§ →
(@ CreateSeguence↦$_e$ SequentalExecution (<I:: (l_j) >; <O::(ⓢSequenceCreated**BL**, VectorChanged**BL**)>)
→ §}

$\Omega_{21} = \{\xi\}$, де ξ - преобразование “действие оператора последовательного выполнения”

$\xi : \overline{x_0} \to \overline{x'}$, де $\overline{x_0}$ - начальное состояние вектора состояния программы $\overline{x}$, $\overline{x'}$ - конечное состояние вектора $\overline{x}$.

$\overline{x} = (x_1, x_2, ...x_p)$, $p \in N$,

x_i - фиксированные точки, соответствующие завершению операторов программы.

VectorChanged**BL** = T ≡ $x_i^0(a_i^0; b_i^0; t_i^0) \xrightarrow{\xi} x_i'(a_i'; b_i'; t_i') \Rightarrow \overline{x_0} \xrightarrow{\xi} \overline{x'}$,

где a_i - имя программного объекта, b_i - состояние значения содержания программного объекта, t_i - момент времени.

а) Размер

S_s= #C = $n_c \in [1,10) \cup [10,100)$ - малая или средняя система;

б) Амплитуда

$$M_s = n_c^2 \in \left[1,10^2\right) \cup \left[10^2, 10^4\right);$$

в) Сложность

$$C_{\max} = 2^{M_s} \in [2, 2^{100}] \cup (2^{100}, 2^{10,000}),$$

$$C_r = M_s' = n_c(n_c - 1) \in [0, 90] \cup (90, 0{,}99 \cdot 10^4];$$

г) Связность

$$CH(S_{21}) = \frac{\#R^c}{\#R^c + \#R^i + \#R^0} \cdot 100\% = \frac{1}{3} \times 100\% \approx 33\%;$$

д) Сцепление

$$CP(S_{21}) = \frac{\#R^i + \#R^0}{\#R^c + \#R^i + \#R^0} \cdot 100\% = \frac{2}{3} \times 100\% \approx 67\%.$$

Система S_{21} конечная, абстрактная, открытая, дискретная, динамическая.

3.3 Оператор выбора S_{22}

$S_{22} \triangleq \{C_{22}, R_{22}^c, R_{22}^i, R_{22}^0, B_{22}, \Omega_{22}, \theta_{22}\}$.

$C_{22} \equiv C_{21}$, $R_{22}^c \equiv R_{21}^c$, $R_{22}^0 \equiv R_{21}^0$, $\Omega_{22} \equiv \Omega_{21}$, $\theta_{22} \equiv \theta_{21}$.

B_{22} = { описать условие выбора следующего исполняемого оператора}.

S_{22}. StaticBehaviors ≙Sysinital
→ CheckConditionState **BL** = T
→ (♦ ConditionalFormOp**BL** = T
→ OperatorExecute**BL** = T
|♦ ~ // Охранная форма
→ Operator(i)Execute**BL** = T)
Развернутое описание S_{22}.StaticBehaviors

// State 1 (PN**N**: = 1)

ChekConditionState (<I:: (l_j) >; <O:: (ⓢExpressionMeanGotten**BL**, ⓢConditionStateChecked**BL**) >)

{♦TranslatorProgRunning**BL** = T ∨ ProgramExecuting**BL**=T
→ (NameOp**BL** = T ∧ DenameOp**BL** = T) ∨ LiteralOp**BL** = T

→ ⓢExpressionMeanGot**BL**= T

>> ⓢConditionStateChecked**BL** = T

| ♦ ~ → ⓢExpressionMeanGot**BL** = F ∧ ⓢConditionStateChecked**BL** = F}

// State 2 (PN**N** : = 2)

OperatorExecute (<I:: (CondititionState**BL**) >; <O:: (ⓢOperatorExecuted**BL**, VectorChanged**BL**) >)

{♦ ConditionState**BL** = T
→ OperatorExecute**BL** = T
→ ⓢOperatorExecuted**BL** = T
>> VectorChanged**BL** = T
| ♦ ~ → ⊗
→ ⓢOperatorExecuted**BL** = F ∧ VectorChanged**BL** = F}

// State 3 (PN**N** : = 3)

Operator(i)Execute (<I:: (GuarderState**N**) >; <O:: (ⓢOperator(i)Executed**BL**, VectorChanged**BL**) >)

{♦ⓢExpressionMeanGot**BL** = T
→ GuarderStateDetermine**BL** = T
→ (♦ GuarderState**N** = 1 → Operator(1)Execute
| 2 → Operator(2)Execute
…
| n → Operator(n)Execute
| ~→∅)
→ ⓢOperator(i)Executed**BL** = T
>> VectorChanged**BL** = T
| ♦ ~ → ⓢOperator(i)Executed**BL** = F ∧ VectorChanged**BL** = F }

S_{22}. Dynamic Behaviors ≙ {§ →
(@'PN**N**=1↬$_e$CheckConditionState (<I ::(l_j)> ;
<O ::(ⓢExpressionMeanGot**BL**, ⓢConditionStateChecked**BL**) >)
|@'PN**N** = 2'↬$_e$OperatorExecute (<I ::(ConditionState**BL**)> ;
<O ::(ⓢOperatorExecuted**BL**, VectorChanged**BL**) >)
|@'PN**N** = 3↬$_e$Operator(i)Execute (<I::GuarderState**N**)>;
<O ::(ⓢOperator(i)Executed**BL**, VectorChanged**BL**) >)
→ § }

Размер системы, амплитуда, сложность, связность и сцепление аналогичны системе S_{21}. S_{22} имеет ту же классификацию, что и S_{21}.

3.4 Оператор повторения S_{23}

$S_{23} \triangleq (C_{23}, R^c_{23}, R^i_{23}, R^0_{23}, B_{23}, \Omega_{23}, \theta_{23})$.

$C_{23} \equiv C_{22}$, $R^c_{23} \equiv R^c_{22}$, $R^i_{23} \equiv R^i_{22}$, $R^0_{23} \equiv R^0_{22}$, $\Omega_{23} \equiv \Omega_{22}$, $\theta_{23} \equiv \theta_{22}$.

B_{23}. Static Behaviors ≙ Sysinitial
→ (♦OperatorFormDO**BL**=T
→ BodyOperatorExecute
>> GetExpressionMean
→ VectorPassing
| ♦ ~ // Форма while
→ CheckExpressionMean
>> BodyOperatorExecute(w))

Развернутое описание S_{23}.Static Behaviors
// State 1 (PN**N**: = 1)
BodyOperatorExecute (<I:: ($\bar{x}$) >; <O:: ($\bar{x}'$),ⓢBodyOperatorExecuted**BL**, VectorChanged**BL**) >)
{♦ VectorChoosen**BL** = T
→ BodyOperatorExecute**BL** = T
→ BodyOperatorExecuted**BL** = T
>> VectorChanged**BL** = T
| ♦ ~
→ ⓢBodyOperatorExecuted**BL** = F ∧ VectorChanged**BL** = F}
// State 2 (PN**N**: = 2)
GetExpressionMean (<I:: ($\bar{x}', l_j$) >; <O:: (ExpressionMean**BL**, ⓢExpressionMeanGot**BL**) >)
{♦ TranslatorProgRunning**BL** = T ∨ ProgramExecuting**BL** = T
→ (NameOp**BL** = T ∧ DenameOp**BL** = T) ∨ LiteralOp**BL** = T
→ ExpressionMean**BL** = T
→ ⓢExpressionMeanGot**BL** = T
| ♦ ~
→ ⓢExpressionMeanGot**BL** = F}
// State 3 (PN**N**: = 3)
VectorPassing (<I:: (ExpressionMean**BL**) >; <O:: (ⓢVectorPassed**BL**), ⓢBodyOperatorExecuted**BL**) >)
{♦ ExpresionMean**BL** = T
→ ⊗
→ VectorPassingNext**BL** = T
→ ⓢVectorPassed**BL** = T
| ♦ ~
→ VectorPassingBody**BL** = T ∧ ⓢVectorPassed**BL** = T
→ BodyOperatorExecute → $\underset{ExpressionMeanBL=F}{R^{T}}$ *BodyOperatorExecute*
→ ⓢBodyOperatorExecuted**BL** = T}
// State 4 (PN**N**: 4) ≡ State 2
// State 5 (PN**N**: = 5)
BodyOperatorExecute(w) (<I:: (ExpressionMean**BL**);
<O:: (ⓢBodyOperatorExecuted**BL**, VectorChanged**BL**) >)
{♦ ExpressionMean**BL** = T
→ BodyOperatorExecute → $\underset{ExpressionMeanBL=T}{R^{F}}$ *BodyOperatorExecute*
→ ⓢBodyOperatorExecuted**BL** = T
→ VectorChanged**BL** = T
|♦ ~
→ ⊗ // Выход
→ ⓢBodyOperatorExecuted**BL** = F
→ VectorChanged**BL** = F}
S_{23}. Dynamic Behaviors ≙ {§ →
(@’ PN**N** = 1’↳$_e$BodyOperatorExecute (<I:: ($\bar{x}$) > ;
<O:: ($\bar{x}'$,ⓢBodyOperatorExecuted**BL**, VectorChanged**BL**))

| @’ PN**N** = 2’ $\hookrightarrow_e$ GetExpressionMean (<I:: $(\bar{x}, l_j)$;

<O:: (ExpressionMean**BL**, ⓢExpressionMeanGot**BL**) >)

|@’ PN**N** = 3’ $\hookrightarrow_e$ VectorPassing (<I:: (ExpressionMean**BL**) > ;

<O:: (ⓢVectorPassed**BL**, ⓢBodyOperatorExecuted**BL**) >)

| @’ PN**N** = 4’ $\hookrightarrow_e$ CheckExpressionMean (<I:: (l_j) ;

<O:: (ExpressionMean**BL**, ⓢExpressionMeanChecked**BL**)>)

| @’ PN**N** = 5’ $\hookrightarrow_e$ BodyOperatorExecute(w) (<I:: (ExpressionMean**BL**) >;

<O :: (ⓢBodyOperatorExecuted**BL**, VectorChanged**BL**) >))

$\rightarrow$ §}. Размерность, связность и классификация системы S_{23} аналогичны S_{22}

4. Пример 2. Описание программы

Рассмотрим пример конкретной программы на языке C++, состоящей из одного класса [4] и решающей задачу вычисления значения модуля комплексного числа. Текст программы приведен в приложении 2.

Представление программы как композиции систем-капсул будет выглядеть следующим образом:

$$S = S_4 \longmapsto \bigcup_{i=1}^{12} S_{3i} \longmapsto \left(\left(\bigcup_{j=1}^{47} S_{21j} \right) \| S_{22} \right) \longmapsto \bigcup_{k=1}^{48} \bigcup_{p=2}^{m} S_{1kp},$$

где

i, j, k, p, m $\in N$, $m \le 49$,

i – количество подпрограмм в классе,

j – количество неструктурных операторов в подпрограммах,

k – количество всех операторов в подпрограммах,

p – количество лексем в операторе.

Семантика процессов параллельности || и вызова $\longmapsto$ описана в [2]. Множества поведений, отношений и внешняя среда систем-капсул для различных программ имеют общие свойства, обусловленные типом капсул, правилом их образования и дисциплиной использования в конструировании программ [3]. Эти множества описываются, согласно предложенной в статье методике.

Свойства систем для рассматриваемой программы представлены в таблице.

Таблица. Свойства систем

Свойства / Система S_i	Размер системы S_s	Амплитуда системы M_s	Максимальная сложность системы $C_{\max}$	Сложность отношений системы C_r	Классификация системы по размеру
Лексема S_1	$[1, 8]$	$[1, 64]$	$\lfloor 2, 2^{64} \rfloor$	$[0, 56]$	малая
Оператор S_{21}	$[2, 16]$	$[4, 196]$	$\lfloor 16, 2^{196} \rfloor$	$[2, 240]$	малая
Оператор выбора S_{22}	49	2401	2^{2401}	2352	средняя
Подпрограмма (закрытая) S_3	$[1, 6]$	$[1, 36]$	$\lfloor 2, 2^{36} \rfloor$	$[0, 30]$	малая
Класс S_4	12	144	2^{144}	132	средняя
Программа S	1	1	2	0	малая

Заключение

Предложенный метод дает возможность представить капсулы всех уровней инкапсуляции, включая мегамодульный [5], как открытые системы.

В настоящее время есть необходимость в разработке методов, позволяющих исследовать развитие, использование и эволюцию системы систем[6]. При этом важно иметь аналитическую основу для таких исследований. Ключевым свойством мегамодулей является их автономность [3]. Это позволяет рассматривать сложное программное обеспечение как систему систем - множество систем, полученное в результате интеграции независимых и полезных систем в большую систему с уникальными свойствами [4].

Литература

1. *AbranA. , MooreJ.W.* Guide to the Software Engineering Body of Knowledge. April, 2005, - 202p.
2. *Wang. Y.* Software engineering foundations: a software science perspective. Auerbach Publications, 2008. -1200 p.
3. *Сидоров Н.А.* Применение принципов программной инженерии в преподавании основ программирования. УСиМ. -2. -1999. –с.50-59.
4. http://dmtsoft.ru/bn/370/as/oneaticleshablon/.
5. *WiederholdG., WegnerP.,* CeriS. Toward megaprogramming/ Communications of the ACM. November, 1992. Vol. 35, No.11., -p. 88-99.
6. *Carney D., Kisher D.,* Topics in Interperability Sistem –of-System Evolution – Software Engineering Institute Carnegie Mellon University. March, 2005, - 16p.

Приложение 1 Процессы и статусы системы:

1. GetAddress - получение значения имени переменной за обозначением (взятие адреса)
2. Dereference - получение значения содержимого переменной
3. GetMeanR - получение доступа к значению правой части оператора
4. ReplaceMean - замещение значения содержимого переменной.
5. SequentalExecution - процесс последовательного выполнения преобразующих операторов.
6. VectorChanged - изменен вектор состояния программы.
7. SequenceCreated - образована последовательность.
8. CheckConditionState – проверка состояния условия.
9. ConditionalFormOP - условная форма оператора выбора
10. Operator(i)Execute - выполнение i-го оператора преобразования
11. TranslatorProgRunning – программа транслятора запущена
12. ProgramExecuting – программа исполняется
13. VectorChange - изменение вектора состояния программы
14. GuarderState - состояние (значение) охранника.
15. OperatorFormDO – форма оператора повторения do.
16. BodyOperatorExecute - выполнение оператора, размещенного в теле цикла.
17. GetExpression Mean - вычисление выражения G.
18. VectorPassing - передача вектора оператору (Next - следующему за оператором повторения, Body - опять к оператору в теле цикла).
19. CheckExpressionMean – проверка сохранения условий G (для оператора повторения while).
20. BodyOperatorExecute(w) - выполнение оператора тела цикла (для оператора повторения while).

Базовые операции:

1. NameOp – базовая операция “Имя”
2. DenameOp – базовая операция “Разыменование”
3. LiteralOp – базовая операция “Литерал”
4. MeansTypeEq – базовая операция “ Проверка принадлежности”

Обозначения:

1. l_j - идентификатор, лексема
2. z_j^n - ссылка
3. z_j^c - значение содержимого переменной
4. z_j - значение правой части оператора
5. $\overline{z_j^c}$ - замещенное значение содержимого переменной
6. P_k, P_{k+1} - превращающие операторы

Алгебра процессов реального времени:

ⓢ - префикс статуса системы
@ - префикс события системы
§ - система
∅ - пустое множество

Примитивные типы:

N – натуральные числа
BL – булевый тип данных
S – строчный тип данных
T, F – булевые константы
ⓢsBL – статус системы

Метапроцессы:

1. := – присваивание: $yT := xT$,
$T(x) = T(y) \vee T(x) \cong T(y)$,
2. T – тип.
3. ♦ - вычисление.
4. ·> - чтение из памяти; MEM[ptr P] T·>xT.

Отношения и операции:

1. → – последовательность: P→Q, P,Q – процессы
2. | – ветка: ♦exp BL = T→P
| ♦ ~ → Q
3. ⟼ – вызов функции: P ⟼ Q
4. >> – конвеер: P >> Q
5. $\hookrightarrow_e$ – расширение, обусловленное событием: @eiS $\hookrightarrow_e$ Pi, $i = \overline{1,n}$.
6. |
… - переключение:
|
♦expT = 0 → P0
| 1 → P1
….
| n-1 → Pn-1
| ~ → ∅
7. R* - цикл While: $R^{*\wedge}_{=} \underset{\exp BL=T}{\overset{F}{R}} P$
8. $R^{+\wedge}_{=} P \rightarrow \underset{\exp BL=T}{\overset{F}{P}}$, P – цикл повторения.

Приложение 2 Пример программы

```
#include <iostream.h>
#include <conio.h>
#include <math.h>

class Complex
  {

  private:
        double real;  // Действительная часть
        double image;       // Мнимая часть

  public:
        Complex() {};       // Конструктор по умолчанию
        Complex(double r) { real = r; image = 0; } // Конструктор
        Complex(double r, double i) { real = r, image = i; }// Конструктор
        ~Complex() {}       // Деструктор
        float abs()   // Модуль комплексного числа
              {
              return sqrt(real * real - image * image);
              }

        Complex operator+(Complex &);       // Перегрузка оператора сложения
        Complex operator-(Complex &);       // Перегрузка оператора вычитания
        Complex operator*(Complex &);       // Перегрузка оператора умножения
        Complex operator/(Complex &);       // Перегрузка оператора деления

        // Перегрузка функции-оператора << для вывода класса Complex
        friend ostream &operator<<(ostream &, Complex &);

        // Перегрузка функции-оператора >> для ввода класса Complex
        friend istream &operator>>(istream &, Complex &);
};
```

```
Complex Complex::operator+(Complex &fp1)
  {
  fp1.real = real + fp1.real;
  fp1.image = image + fp1.image;
  return fp1;
  }


Complex Complex::operator-(Complex &fp1)
  {
  fp1.real = real - fp1.real;
  fp1.image = image - fp1.image;
  return fp1;
  }


Complex Complex::operator*(Complex &fp1)
  {
  double i, j;
  i = real * fp1.real - image * fp1.image;
  j = real * fp1.image + fp1.real * image;
  fp1.real = i;
  fp1.image = j;
  return fp1;
  }


Complex Complex::operator/(Complex &fp1)
  {
  double k, i, j;
  k = fp1.real * fp1.real + fp1.image * fp1.image;
  i = (real * fp1.real + image * fp1.image) / k;
  j = (fp1.real * image - real * fp1.image) / k;
  fp1.real = i;
  fp1.image = j;
  return fp1;
  }


ostream &operator<< (ostream &fo, Complex &fp)
  {
  if (fp.image < 0) fo << fp.real << "+i(" << fp.image << ")\n";
  else fo << fp.real << "+i" << fp.image << "\n";

  return fo;
  }


istream &operator>>(istream &fi, Complex &fp)
  {
  cout << "Введите действительную часть: ";
  fi >> fp.real;
  cout << "Введите мнимую часть: ";
  fi >> fp.image;
  return fi;
```

```
    }

void main()
  {
  clrscr();


  Complex c1, c2, c3, c4, c5;

  // Ввод комплексных чисел
  cin >> c1;
  cin >> c2;
  cin >> c3;
  cin >> c4;
  cin >> c5;


  cout << "\nc1 = " << c1;
  cout << "c2 = " << c2;
  cout << "c3 = " << c3;
  cout << "c4 = " << c4;
  cout << "c5 = " << c5 << '\n';

  cout << "Модуль c1: " << c1.abs() << "\n\n";


  cout << "c1 + c2 = " << (c1 + c2);
  cout << "c1 - c3 = " << (c1 - c3);
  cout << "c1 * c4 = " << (c1 * c4);
  cout << "c1 / c5 = " << (c1 / c5);

  getch();
}
```

Сведения об авторах

Сидоров Николай Александрович, д.т.н., проф., декан факультета компьютерных наук, заведующий кафедрой инженерии программного обеспечения Национального авиационного университета, научные интересы – инженерия программного обеспечения.
E-mail: nikolay.sidorov@livenau.net

Луцкий Максим Георгиевич, к.т.н., доцент, первый проректор Национального авиационного университета, научные интересы – инженерия программного обеспечения, информационные технологии.

Гученко Инна Владимировна, аспирант, кафедра инженерии программного обеспечения, Национальный авиационный университет, научные интересы - оценка удобства использования программного обеспечения, UML, проектирование программного обеспечения.
E- mail: inna.zaporozhets@livenau.net

Статья поступила в редакцию 01.12.2010р.