Степанян Карлен Багратович

ФОРМАЛИЗАЦИЯ ДВУМЕРНЫХ ГРАФО-ПОДОБНЫХ ДИАГРАММ

Аннотация

В статье излагается формализм для описания двумерных графо-подобных диаграмм. Основанный на формализации диаграмм, предложенной Е.А. Жоголевым, он расширяет ее, вводя конкретные типы двумерных конструкций и допустимые отношения между ними. При этом рассматриваются элементы таких визуальных языков, как сети Петри, ERD, UML, IDEFx и SDL.

Ключевые слова: графо-подобная диаграмма, формализация графических конструкций, нотация, конкретный синтаксис.

ВВЕДЕНИЕ

Для формализации двумерных графо-подобных диаграмм (далее просто диаграмм) Е.А. Жоголев [1] выделил два множества конструкций на диаграмме (буквенное обозначение выбрано автором):

V – множество двумерных конструкций, которые располагаются на диаграмме контекстно-независимо (вершины);

E – множество линий, которые располагаются на диаграмме в зависимости от расположения инцидентных им двумерных конструкций (ребра).

Данного представления достаточно для формализации понятия двумерной графо-подобной диаграммы в первом приближении, однако это слишком абстрактный формализм для описания нотации реальных диаграмм во всех деталях. В этой статье предлагается расширенный формализм, достаточный для описания нотации современных диаграмм. Технические аспекты отображения, такие как толщина линий, цвета, стили, шрифты и т. д., опускаются, поскольку они несущественны.

Все конструкции, которые могут присутствовать на диаграмме, предлагается разбить на следующие множества:

– множество *меток* T – множество конструкций, отображающих текст;

– множество *декораций* $D \subset V$ – множество конструкций, предназначенных для отображения статических элементов, например, таких как значки на концах линии;

– множество *фигур* $F \subset V$ – множество конструкций, отображающих сущности диаграммы;

– множество *составных фигур* $CF \subset F$ – фигуры, которые содержат в себе другие конструкции для составления сложной геометрической композиции, представляющей на диаграмме одну сущность;

– множество *контейнеров* $K \subset F$ – фигуры, которые могут содержать в себе другие конструкции, представляющие на диаграмме отдельные сущности;

– множество *рамок* $Fr \subset F$ – фигуры, которые всегда отображаются поверх других вершин и предназначены для отображения дополнительной семантики с помощью обрамления других конструкций;

– множество *ребер* E – множество контекстно-зависимых конструкций, отображающих отношения между сущностями.

В формализации Жоголева ребра не имеют направления, однако для описания нотации большинства языков есть необходимость отличать полюс-начало ребра от полюса-конца, поэтому, не умаляя общности, мы будем считать, что все ребра направленные.

ОБЩИЕ ОПРЕДЕЛЕНИЯ

Любое представление вершины на диаграмме можно разбить на комбинацию из конечного числа геометрических примитивов. В качестве примитивов, из которых строятся двумерные конструкции, предлагается взять следующее множество *G*:

– прямоугольник;
– эллипс;
– прямоугольник со скругленными углами;
– ломаная;
– замкнутая ломаная.

Для *V* определено отношение $geom : V \to G$, сопоставляющее каждой вершине геометрический примитив из множества *G*.

Не умаляя общности, будем считать, что ребро представляется всегда ломаной линией.

Введем обозначения:

C – множество всех конструкций диаграммы;

N – множество натуральных чисел;

R – множество действительных чисел;

Z – множество чисел, которые используются для представления координат или размеров элементов диаграммы;

$Point = Z \times Z$ – множество точек плоскости, на которой отображается диаграмма;

$point : V \to Point$ – сопоставляет любой вершине диаграммы координаты ее левого верхнего угла, эти координаты будем также называть координатами вершины;

$Size = Z^{+} \times Z^{+}$ – множество всевозможных размеров элементов по ширине и высоте;

$size : V \to Size$ – сопоставляет любой вершине ее размер на диаграмме.

Для определения положения и размеров геометрических конструкций используется прямоугольная система координат. Поскольку любая диаграмма имеет конченые размеры, множество *Z* также конечно, но достаточно велико для представления любых реальных диаграмм.

На строго формализованной диаграмме каждая конструкция имеет свой *тип*. Например, в сетях Петри [6] все вершины поделены на два типа – позиции и переходы. Оператор $type : C \to Type$ сопоставляет конструкции диаграммы ее тип. У любой конструкции определен единственный тип:

$$\forall c : c \in C \to \exists ! \, type(c) \in Type .$$

Обозначим множество всех типов вершин на диаграмме как

$$Type = \{ type(v) \}_{v \in V} .$$

Определим множество *операторов индексации* для произвольного множества *A* как

$$Index(A) = \{ index \mid index : A \to N \text{ – инъективна}, \ \forall_{a \in A} \exists i : i = index_i(a) \},$$

Тогда следующее высказывание обозначает, что множество *A* индексируемо оператором *i*

$$enumerable(A, i) \equiv i \in Index(A) .$$

Следующее выражение определяет равенство двух множеств конструкций диаграммы с точностью до типов:

$$eqtype(A_1, A_2) \equiv A_1 \in C \wedge A_2 \in C \wedge |A_1| = |A_2| \wedge \exists i_1 : enumerable(A_1, i_1) \wedge \exists i_2 : enumerable(A_2, i_2) \wedge (\forall a_1 \in A_1 \forall a_2 \in A_2 : i_1(a_1) = i_2(a_2) \rightarrow type(a_1) = type(a_2)),$$

то есть два множества конструкций равны с точностью до типов, когда они равны по мощности и типы элементов с одинаковыми индексами у них равны.

МЕТКИ

Метки – это конструкции, отображающие на диаграмме текст. Ни одна диаграмма в данный момент не обходится без меток. Например, в сетях Петри, диаграмме состояний и т. д. метки идентифицируют вершины на диаграмме. Обобщая, можно утверждать, что как вершины, так и ребра диаграммы могут иметь связанные с ними метки.

Обозначим T – множество меток (текстовых элементов) на диаграмме. Метки не являются ни вершинами, ни ребрами, поэтому множество всех конструкций на диаграмме – это объединение трех множеств:

$$C = V \cup E \cup T .$$

Метки являются контекстно-зависимыми элементами – их расположение зависит от расположения родительского элемента. Однако простого отображения метки рядом (или внутри) родительского элемента в некоторых случаях недостаточно, так как расположение метки может носить семантическую составляющую. Например, ассоциация из диаграммы классов UML [2] может иметь название, которое должно выводиться в середине линии, и метки на своих полюсах, которые должны располагаться рядом с соответствующим полюсом. Это означает, что для каждой метки должно существовать правило ее отображения относительно расположения родительского элемента на диаграмме.

Контекстная зависимость для метки t от конструкции p может быть определена через отношение *прикрепления*:

$$attachedt(p, t, op, x) \equiv p \in V \cup E \wedge t \in T \wedge point(t) = op(p, t, x),$$

то есть метка является контекстно-зависимой в том случае, если ее координаты на диаграмме определяет оператор прикрепления

$$op : V \cup E \times T \times AttCont \rightarrow Point$$

с помощью контекста прикрепления $x \in AttCont$.

Основываясь на отношении прикрепления одной метки, можно определить множество всех меток, прикрепленных к конструкции, как

$$attachedtS(p, S, op, X) \equiv \forall s \in S \; \exists! x \in X : attachedt(p, s, op, x) \wedge \neg \exists op, x, s \in T \setminus S : attachedt(p, s, op, x).$$

Поскольку у всех конструкций одинакового типа одинаковое количество прикрепленных меток и они располагаются одинаково, то их множества контекстов прикрепления равны, а множества прикрепленных меток равны с точностью до типов:

$$type(p_1) = type(p_2) \wedge attachedtS(p_1, S_1, op, X_1) \wedge attachedtS(p_2, S_2, op, X_2) \rightarrow X_1 = X_2 \wedge eqtype(S_1, S_2).$$

ОПЕРАТОР ПРИКРЕПЛЕНИЯ МЕТКИ К ВЕРШИНЕ

Для всех вершин диаграммы одинакового типа прикрепление метки должно выглядеть одинаково. Но вершины могут иметь разное положение и размеры, поэтому кон-

текст прикрепления определяется точками в относительных системах координат с осями, нормированными по размеру вершины. В такой системе точка (0, 0) соответствует левому верхнему углу, а точка (1, 1) соответствует правому нижнему углу элемента. При этом метка рассматривается также как двумерная конструкция со своими размерами и расположением.

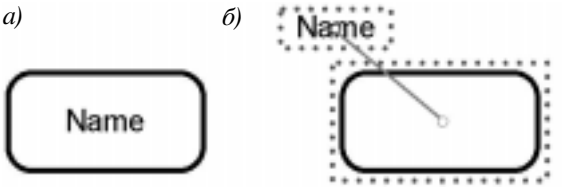

Рис. 1: *а)* вершина с прикрепленной в центре меткой; *б)* схематическое изображение прикрепления цетра метки к центру вершины

На рис. 1*а* приведен пример вершины с прикрепленной к центру меткой. Для того чтобы описать подобное прикрепление, необходимо указать, что центр метки должен совпадать с центром вершины (см. рис. 1*б*). Центру метки соответствует точка (0.5, 0.5) в относительной системе координат метки. Для вершины точка прикрепления определяется аналогично.

Формально контекстом прикрепления для оператора являются следующие параметры:

$r_v = (rx_v, ry_v) \in R \times R$ – точка прикрепления родительской вершины в ее системе координат;

$r_t = (rx_t, ry_t) \in R \times R$ – точка прикрепления метки в системе координат метки.

Тогда для любой вершины v, имеющей координаты $(x_v, y_v) = point(v)$ и размеры $(w_v, h_v) = size(v)$, оператор прикрепления TAV вычисляет положение прикрепленной метки по ее размерам:

$$TAV(v, r_t) = \begin{pmatrix} -rx_t & 0 & rx_v \cdot w_v + x_v \\ 0 & -ry_t & ry_v \cdot w_v + y_v \\ 0 & 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} x_t \\ y_t \\ 1 \end{pmatrix} = TAV(v, r_t) \times \begin{pmatrix} w_t \\ h_t \\ 1 \end{pmatrix},$$

где $(w_t, h_t) \in Size$ – размер метки, вычисленный по тексту, который необходимо отобразить; $(x_t, y_t) \in Point$ – вычисленное положение метки на диаграмме.

ОПЕРАТОР ПРИКРЕПЛЕНИЯ МЕТКИ К РЕБРУ

Предполагается, что для линий, представляющих ребра на диаграмме, определены следующие функции:

$length : E \to Z^+$ – функция вычисления длины;

$apoint : E \times R \to Point$ – функция, определяющая точки на линии по относительному положению вдоль нее;

$angle : E \times Point \to R$ – функция, определяющая угол поворота линии по часовой стрелке относительно оси x в любой ее точке S.

Прикрепление метки к ребру отличается от прикрепления метки к вершине тем, что для указания точки прикрепления на ребре используется одна координата, обозначающая положение вдоль ребра относительно его длины. То есть 0 означает полюс-начало ребра, а точка 1 означает полюс-конец ребра. Контекстом для оператора прикрепления метки к ребру являются следующие параметры:

$r_e \in R$ – относительное положение точки прикрепления на ребре;

$r_t = (rx_t, ry_t) \in R \times R$ – точка прикрепления метки в системе координат метки.

Пусть $e \in E$ – рассматриваемое ребро, тогда

$(x_e, y_e) = apoint(e, r_e)$ – точка прикрепления на ребре в координатах плоскости,

$\alpha = angle(e, (x_e, y_e))$ – угол поворота линии в точке прикрепления.

Прикрепление метки к ребру указывается в предположении, что ребро отображается горизонтально слева на право. При повороте ребра прикрепленные метки поворачиваться не должны, поэтому поворот ребра необходимо учитывать при расчете точки прикрепления у метки (см. рис. 2). Для этого необходимо повернуть точку прикрепления в относительных координатах вокруг центра метки на угол α:

$$\begin{pmatrix} 1 & 0 & 0.5 \\ 0 & 1 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & -0.5 \\ 0 & 1 & -0.5 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} rx_t \\ ry_t \\ 1 \end{pmatrix} =$$
$$= \begin{pmatrix} rx_t \cos\alpha - ry_t \sin\alpha + 0.5(\sin\alpha - \cos\alpha + 1) \\ rx_t \sin\alpha + ry_t \cos\alpha - 0.5(\sin\alpha + \cos\alpha - 1) \\ 1 \end{pmatrix}.$$

Обозначим

$\overline{rx_t} = rx_t \cos\alpha - ry_t \sin\alpha + 0.5(\sin\alpha - \cos\alpha + 1)$,

$\overline{ry_t} = rx_t \sin\alpha + ry_t \cos\alpha - 0.5(\sin\alpha + \cos\alpha - 1)$.

Тогда для ребра e оператор прикрепления TAE вычисляет положение прикрепленной метки по ее размерам:

$$TAE(e, r_t) = \begin{pmatrix} -\overline{rx_t} & 0 & x_e \\ 0 & -\overline{ry_t} & y_e \\ 0 & 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} x_t \\ y_t \\ 1 \end{pmatrix} = TAE(e, r_t) \times \begin{pmatrix} w_t \\ h_t \\ 1 \end{pmatrix},$$

где $(w_t, h_t) \in Size$ – размер метки, вычисленный по тексту, который необходимо отобразить; $(x_t, y_t) \in Point$ – вычисленное положение метки на диаграмме.

На рис. 2 *а* приведен пример ребра с прикрепленными к концу декорацией (см. ниже) и меткой. Для прикрепления метки необходимо указать, что ее правый нижний угол должен совпадать с концом горизонтально расположенного ребра (см. рис. 2 *б*). На рис. 2 *в* изображено то же ребро, но повернутое на 90 градусов против часовой стрелки. Для учета поворота ребра точка прикрепления у метки меняет свое положение, что отображено на рис. 2 *г*.

ДЕКОРАЦИЯ

Декорация – это статичная двумерная конструкция, которая может быть прикреплена к другим конструкциям диаграммы. Статичность здесь понимается как отсутствие возможности у пользователя менять параметры декораций, заданные при описании нотации (например ее размер). Самый простой и яркий пример использования декора-

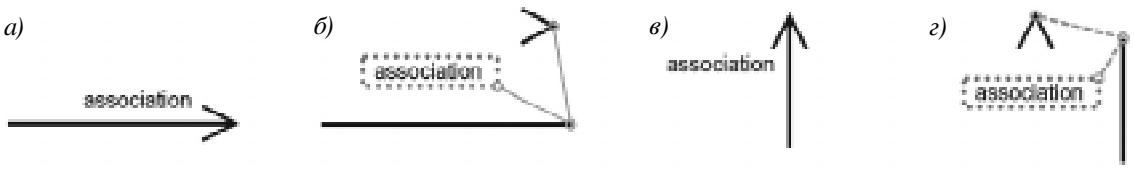

Рис. 2: *а)* ребро с прикрепленными к концу меткой и декорацией;
б) схематическое изображение прикрепления метки и декорации;
в) ребро, повернутое на 90 градусов против часовой стрелки;
г) схематичное изображение конструкций и точек прикрепления после поворота

ции – это указание направления у ребра (см. рис. 2). Для этого обычно используются разнообразные стрелки на соответствующем полюсе. Некоторые визуальные языки используют более сложные декорации для отображения дополнительной семантики, например, нотация Crow's Foot диаграмм сущность-отношение (Entity-Relation Diagram – ERD) [7] использует различные наборы из окружности и отрезков для отображения кратности отношения.

Декорация является вершиной и может представлять на диаграмме отдельную сущность, то есть может иметь инцидентные ребра. Например, на диаграммах состояний UML и диаграммах деятельности UML черный круг обозначает начальное состояние, а ромб ветвление. Пользователь может изменять их положение, инцидентные им ребра, но не может изменять размер.

Формально множество вершин V можно разбить на два непересекающихся подмножества:

$$V = D \cup F\ ,\ D \cap F = \varnothing ,$$

где D – множество декораций и F – множество фигур.

Для элементов множества D верно

$$\forall d_1, d_2 \in D : type(d_1) = type(d_2) \rightarrow size(d_1) = size(d_2) ,$$

то есть у любых двух декораций одинакового типа одинаковые размеры.

Как было отмечено выше, декорация может быть прикреплена к другим конструкциям так же, как и метка. Отношение прикрепления для нее определяется абсолютно аналогичным образом:

$$\forall d_1, d_2 \in D : type(d_1) = type(d_2) \rightarrow size(d_1) = size(d_2) .$$

Аналогично определяется множество всех прикрепленных к конструкции декораций:

$$attacheddS(p, S, op, X) \equiv \forall s \in S\ \exists! x \in X : attachedd(p, s, op, x)$$
$$\wedge \neg \exists op, x, s \in D \setminus S : attachedd(p, s, op, x)$$

Поскольку у всех конструкций одинакового типа одинаковое количество прикрепленных декораций и они располагаются одинаково, то их множества контекстов прикрепления равны, а множества прикрепленных декораций равны с точностью до типов:

$$type(p_1) = type(p_2) \wedge attacheddS(p_1, S_1, op, X_1) \wedge attacheddS(p_2, S_2, op, X_2) \rightarrow$$
$$\rightarrow D_1 = D_2 \wedge |S_1| = |S_2| \wedge typeequality(S_1, S_2)$$

Оператор прикрепления декорации к вершине абсолютно аналогичен оператору прикрепления метки к вершине, так как декорация, как и метка, обладает размером и положением.

ОПЕРАТОР ПРИКРЕПЛЕНИЯ ДЕКОРАЦИИ К РЕБРУ

Оператор прикрепления декорации к ребру отличается от оператора прикрепления метки к ребру тем, что декорация, в отличие от метки, должна поворачиваться вместе с ребром. Контекстом для оператора прикрепления декорации к ребру являются следующие параметры:

$r_e \in R$ – относительное положение точки прикрепления на ребре,

$r_d = (rx_d, ry_d) \in R \times R$ – точка прикрепления декорации в системе координат декорации.

Пусть

$(x_e, y_e) = apoint(e, r_e)$ – точка прикрепления на ребре в координатах плоскости,

$\alpha = angle(e, (x_e, y_e))$ – угол поворота линии в точке прикрепления.

Тогда оператор прикрепления *DAE* вычисляет положение неповернутой прикрепленной декорации по ее размерам:

$$DAE(e,r_d)=\begin{pmatrix}-rx_d & 0 & x_e\\ 0 & -ry_d & y_e\\ 0 & 0 & 1\end{pmatrix},\qquad \begin{pmatrix}x_d\\ y_d\\ 1\end{pmatrix}=DAE(e,r_d)\times\begin{pmatrix}w_d\\ h_d\\ 1\end{pmatrix},$$

где $(w_d,h_d)\in Size$ – размер декорации, а $(x_d,y_d)\in Point$ – вычисленное положение неповернутой декорации на диаграмме.

После этого всю декорацию необходимо повернуть на угол *б* вокруг точки прикрепления, что делает следующий оператор:

$$ROT(d,r_d)=\begin{pmatrix}1 & 0 & x_d+rx_d\cdot w_d\\ 0 & 1 & y_d+ry_d\cdot h_d\\ 0 & 0 & 1\end{pmatrix}\times\begin{pmatrix}\cos\alpha & -\sin\alpha & 0\\ \sin\alpha & \cos\alpha & 0\\ 0 & 0 & 1\end{pmatrix}\times\begin{pmatrix}1 & 0 & -x_d-rx_d\cdot w_d\\ 0 & 1 & -y_d-ry_d\cdot h_d\\ 0 & 0 & 1\end{pmatrix}.$$

Пример, иллюстрирующий прикрепленную декорацию к ребру, представлен на рис. 2 *а*. Декорация крепится в точке, соответствующей середине правой стороны, к концу ребра. После поворота ребра (рис. 2 *в*) декорация поворачивается вместе с ним.

ФИГУРА

Основной конструкцией для представления вершин на диаграмме является *фигура*. В отличие от декорации, фигуры не могут быть прикреплены к другим конструкциям, фигуры одного типа могут иметь разные размеры и могут содержать в себе другие конструкции (см. *сложная фигура* и *контейнер*). Примерами фигур могут быть состояние из диаграммы состояний, сущность и отношение из ERD, модуль поведения из IDEF3 [5] и т.д. Для двух фигур одного типа, верно, что они представлены на диаграмме одинаковым геометрическим примитивом:

$$f_1\in F\wedge f_2\in F\wedge type(f_1)=type(f_2)\rightarrow geom(f_1)=geom(f_2).$$

Далее рассмотрены два подмножества фигур – множество составных фигур и контейнеров, которые образуют сложные конструкции.

СОСТАВНАЯ ФИГУРА

В современных визуальных языках семейства IDEFx [3,4,5] и UML для представления вершин диаграмм простых графических примитивов недостаточно. Например, конструкция для отображения класса на диаграмме классов UML должна обладать вертикальными секциями для отображения названия класса, его атрибутов и его операций. Конструкция, отображающая модуль поведения (unit of behavior) в IDEF3, имеет внизу две горизонтальные секции для отображения порядкового номера элемента и его связи с элементами из IDEF0 [4]. Все перечисленные конструкции представляют на диаграмме одну сущность визуального языка, однако используют сложные геометрические образы. Для описания подобных образов можно использовать комбинации из нескольких контекстно-зависимых вершин. Контекстная зависимость определяет отношение принадлежности зависимой вершины (дочерней) от независимой (родительской). Если дочерние вершины вместе с родительской представляют на диаграмме одну сущность визуального языка, то родительская вершина называется *составной фигурой*. То есть составную фигуру можно описать как фигуру, содержащую в себе конечный заранее определенный набор других конструкций. Конкретное расположение дочерней вер-

шины определяется оператором расположения, связанным с родительской вершиной.

Определим множество операторов расположения на диаграмме:

$$Layout = \{layout \mid layout : V \times V \to Point \times Size\},$$

где оператор *layout* определяет положение и размер на диаграмме дочерней вершины, в зависимости от ее параметров и параметров родительской вершины. Тогда высказывание

$$constrained(p,c,layout) \equiv p,c \in V \wedge layout \in Layout$$
$$\wedge \forall point(p), \forall size(p) : (point(c), size(c)) = layout(p,c)$$

определяет контекстную зависимость p от c и утверждает, что расположение вершины c определяется с помощью оператора *layout*.

На рис. 3 изображена составная фигура, состоящая из трех других фигур (выделены штрихпунктиром), расположенных вертикально друг под другом. Родительская фигура представлена в виде прямоугольника с закругленными углами. Первая дочерняя фигура не представлена геометрическим примитивом, к ее центру прикреплена метка для отображения имени сущности. Вторая дочерняя фигура представлена в виде горизонтальной линии, идущей по верхней грани фигуры и также является составной фигурой (ее дочерние фигуры выделены пунктиром). Третья дочерняя фигура представлена аналогичным геометрическим примитивом, но является контейнером (см. ниже).

Рис. 3. Пример составной фигуры, представляющей составное состояние на диаграмме состояний UML (пунктирными прямоугольниками выделены границы дочерних фигур)

В приведенном выше примере оператор расположения для родительской фигуры может быть формализован следующим образом. Пусть p – родительская фигура, а c_i – дочерние фигуры $i = 0..n$. Тогда

$(x_p, y_p) = point(p)$ – координаты родительской фигуры;

$(w_p, h_p) = size(p)$ – размер родительской фигуры;

$(x_c^i, y_c^i) = point(c_i)$ – вычисляемые координаты i-ой дочерней фигуры;

$(w_c^i, h_c^i) = size(c_i)$ – вычисляемый размер i-ой дочерней фигуры;

тогда координаты и размер i-ой дочерней фигуры вычисляется по следующим формулам:

$$x_c^i = x_p, \quad y_c^i = \begin{cases} y_c^i + h_c^i + y_p, & i > 0 \\ y_p, & i = 0 \end{cases}, \quad w_c^i = w_p,$$

h_c^i – не изменяется оператором.

Ниже описана формализация составной фигуры.

Будем говорить, что дочерняя вершина $c \in V$ принадлежит родительской вершине $p \in V$ в том случае, если расположение c на диаграмме зависит от расположения p. Отношение принадлежности определяется следующим высказыванием:

$$parentv(p,c) \equiv \exists! layout \in Layout : constrained(p,c,layout)$$

Данное отношение обладает рядом свойств (приводятся без доказательств):

$parentv(p,c) \equiv \exists! layout \in Layout : constrained(p,c,layout)$ – оно асимметрично;

$\forall v : \neg\, parentv(v,v)$ – оно антирефлексивно;

$parentv(p_1,c) \wedge parentv(p_2,c) \rightarrow p_1 = p_2$ – у дочерней вершины может быть только одна родительская вершина.

Определим родительскую вершину, содержащую в себе множество дочерних вершин:

$$parentvC(p,C) \equiv C \subset V \wedge C \neq \varnothing \wedge \forall c \in C \rightarrow parentv(p,c).$$

Тогда множество всех дочерних вершин определяется как

$$parentvFC(p,FC) \equiv parentvC(p,FC) \wedge \forall C \subset V : parentvC(p,C) \rightarrow C \subset FC.$$

Обозначим родительскую вершину с единственным оператором расположения как

$$singlelayout(p,layout) \equiv \forall c, \forall l : constrained(p,c,l) \rightarrow l = layout,$$

$$uniquelayout(p) \equiv \exists layout : singlelayout(p,layout),$$

тогда формальное определение *составной фигуры* описывается следующим образом:

$$compfig(cf) \equiv cf \in F \wedge \exists C : parentvFC(cf,C) \wedge uniquelayout(cf).$$

Все составные фигуры одинакового типа содержат множества дочерних конструкций, равных с точностью до типа, что описывается следующим высказыванием:

$$compfig(cf_1) \wedge compfig(cf_2) \wedge type(cf_1) = type(cf_2) \wedge parentvFC(cf_1, FC_1) \wedge parentvFC(cf_2, FC_2) \rightarrow eqtype(FC_1, FC_2).$$

КОНТЕЙНЕР

Многие визуальные языки используют вложенность одних вершин в другие для отображения дополнительной семантики, например, вложенные состояния на диаграмме состояний. Вершины, которые содержат в себе другие вершины и эта вложенность имеет семантическую значимость, называются *контейнерами*. Отличие контейнера от составной фигуры заключается в том, что вершина, принадлежащая контейнеру, отображает связанный с ним, но отдельный экземпляр сущности визуального языка. В то время как вершина, принадлежащая составной фигуре, является частью графической нотации для отображения одного экземпляра сущности. У разных контейнеров может быть разное количество принадлежащих им вершин, однако контейнер содержит вершины только определенного типа – это ограничение накладывается семантикой визуального языка. Следующее высказывание ограничивает множество типов вершин, которые могут принадлежать вершине p:

$$typerestr(p,T) \equiv T \subset Type \wedge \forall v : parentv(p,v) \rightarrow type(v) \in T.$$

Формально контейнер определяется как

$$contr(ct) \equiv ct \in F \wedge \neg compfig(ct) \wedge \exists T : typerestr(ct,T) \wedge \exists v : parentv(ct,v),$$

то есть контейнер – это фигура, которая не может быть одновременно составной фигурой, обладающей хотя бы одной вершиной, и задано множество типов всех вершин, которые могут ей принадлежать. Множества типов всех вершин, которые могут принадлежать двум контейнерам одинакового типа, равны:

$$contr(ct_1) \wedge contr(ct_2) \wedge type(ct_1) = type(ct_2) \wedge typerestr(ct_1,T_1) \wedge typerestr(ct_1,T_1) \rightarrow T_1 = T_2.$$

РАМКА

Наряду с обычными вершинами, в таких языках, как UML, IDEFx, SDL [8] и т. д., широко используется рамка для обрамления других конструкций. Ее основное отличие от других вершин заключается в том, что она всегда отображается поверх них. Рамка не содержит в себе конструкции, которые обрамляет. Сама по себе рамка может быть как простой фигурой (например SDL и UML), так и составной (IDEFx) или даже контейнером (когда у нее неопределенное количество секций, как в языке UML).

Для формализации понятия рамки необходимо ввести оператор, определяющий порядок отображения вершин

$$zorder : V \to N \ .$$

Тогда, если обозначить множество рамок как $Fr \subset F$, будет верно следующее утверждение

$$\forall fr \in Fr \ \ \forall v \in V \setminus Fr : zorder(v) < zorder(fr) \ .$$

В то же время порядок отображения самих рамок определяется таким же образом, как и простых фигур.

РЕБРА

Что касается ребер, формализм следует дополнить определением допустимых пар вершин, которым может быть инцидентно ребро. Поскольку вершины имеют разные типы, то ребро может быть инцидентно только определенным вершинам, что диктуется семантикой сущностей и отношений между ними.

Пусть следующие высказывания определяют, может ли быть ребро e инцидентно вершине v

$source(e,v)$ – своим полюсом-началом;

$target(e,v)$ – своим полюсом-концом.

Тогда следующее высказывание определяет множество типов вершин, которым может быть инцидентно ребро полюсом-началом:

$$sourceT(e,ST) \equiv ST \subset Type \wedge \forall v \in V : type(v) \in ST \to sorce(e,v) \ .$$

Аналогично для полюса-конца

$$targetT(e,TT) \equiv TT \subset Type \wedge \forall v \in V : type(v) \in TT \to target(e,v) \ .$$

ЗАКЛЮЧЕНИЕ

Основываясь на предложенной классификации конструкций диаграммы и выделенных для каждого класса свойствах и отношениях, можно описать нотацию современных диаграмм. Для этого необходимо определить типы конструкций, которые будут присутствовать на диаграмме, для каждого типа задать один из возможных классов и определить в соответствии с классом свойства и отношения, которыми будут обладать конструкции данного типа. Подобный подход лег в основу языка DiaDeL – языка описания диаграмм, рассмотренного в статьях [9–11]. Кроме вышеизложенного, DiaDeL позволяет определять «косметические» свойства представления конструкций, такие как цвет, толщина и тип линий, шрифт для текста, его размер и т. д.

Литература

1. *Жоголев Е.А.* Графические редакторы и графические грамматики // Программирование, 2001. № 3. С. 30–42.

2. Unified Modeling Language (http://www.uml.org/).

3. Integration Definition methods (IDEF) (http://www.idef.com).

4. Integration Definition for Function Modeling (IDEF0) (http://www.idef.com/pdf/idef0.pdf).

5. Integration Definition for Process Description Capture (IDEF3) (http://www.idef.com/pdf/Idef3_fn.pdf).

6. *Котов В.Е.* Сети Петри. М.: Наука, 1984. 160 с.

7. Entity-relationship diagram (ERD) (http://ru.wikipedia.org/wiki/ERD).

8. Specification and Description Language implementer's guide – Version 1.0.2 (http://www.itu.int/rec/T-REC-Z.Imp100-200709-I/en).

9. *Степанян К.Б.* Язык описания диаграмм // Научно-технические ведомости СПбГПУ, 2006. № 6–1. С. 36–41.

10. *Новиков Ф.А., Степанян К.Б.* Язык описания диаграмм // Информационно-управляющие системы, 2007. № 4. С. 28–36.

11. *Степанян К.Б.* Использование языка описания диаграмм // Информационно-управляющие системы, 2009. № 1. С. 25–32.

Abstract

The paper proposes formalism to specify plane graph like diagrams. Formalism is based on formalization described by Zhogolev E. A. New kind of diagrammatic constructions introduced with possible relationships between them. Elements of such visual languages as Petri's Nets, ERD, UML, IDEF and SDL are considered throughout the paper.

Keywords: diagramming, graph-like diagram, notation, drawing formalization.

Степанян Карлен Багратович,
магистр математических наук,
проектировщик программного обеспечения ЗАО «Моби.Деньги»
karlens@hotmail.com