УДК 004(4’242+053)

С.И. Чайников[1], А.С. Солодовников[2]
[1]ХНУРЭ, г. Харьков, Украина;
[2]ХНМУ, г. Харьков, Украина

ФОРМАЛИЗОВАННОЕ ОПИСАНИЕ ГРАФ-МОДЕЛИ ПРЕДМЕТНОЙ ОБЛАСТИ

Рассмотрено формализованное описание граф-модели предметной области и элементов диалоговой системы. Описаны алгоритмы обработки граф-модели, приводящие граф к форме, необходимой для обеспечения функций контроля и управления работой диалоговой системы.

ДИАЛОГОВЫЕ СИСТЕМЫ, ГРАФ-МОДЕЛЬ ПРЕДМЕТНОЙ ОБЛАСТИ, ГРАФ ГЕРЦА, КОНЕЧНЫЙ АВТОМАТ

Введение

Большинство современных информационных систем (ИС) разрабатывается с учетом модульного принципа, что дает ряд преимуществ, например, в выборе языка программирования или локализации мест ошибок при отладке программного средства (ПС). Учитывая модульный принцип, структура ПС легко описывается с использованием теории графов, а функциональные особенности – на основе теории автоматов (как показывает анализ литературы). Также графы применимы при описании предметных областей для ИС. Тем не менее, всегда остается проблема оптимального построения ПС и моделей предметных областей (ПрО). Такое описание должно обеспечивать возможность контроля над выполнением вычислений, позволять видоизменять саму структуру программы и осуществлять сквозное управление данными в ИС. В связи с этим целью работы является формализованное описания граф-модели ПрО и элементов диалоговой системы (ДС), описанной в [1].

1. Формальное описание модели диалоговой системы

Структура ДС описывается с помощью ориентированного графа, приведенного к ярусно-параллельной форме (ЯПФ) [2]. Такая форма позволяет осуществить параллельную организацию вычислений и распределить вычислительные процессы на компьютерных кластерах, к примеру, гетерогенных кластерах, или многопроцессорных системах, относящихся к классам SMP и NUMA. Однако описание структуры программы с помощью канонической ЯПФ приводит к появлению такой проблемы, как одновременная рассылка одних и тех же данных по многим процессорам (или рабочим станциям), что влечет за собой увеличение времени работы ПС. Также появляется проблема организации многопользовательского доступа к общим массивам данных. Прежде чем приступать к решению указанных проблем, необходимо выполнить формальное описание модели ДС на базе орграфа.

Пусть задан граф (рис. 1):

$$G = (V, X), \quad (1)$$

где V – множество вершин v; X – множество дуг $x_{ij}=(v_i,v_j)$ графа G, соединяющих вершины v_i и v_j между собой, для которых задано направление ($i, j = \overline{1,n}$, где n – количество вершин графа G).

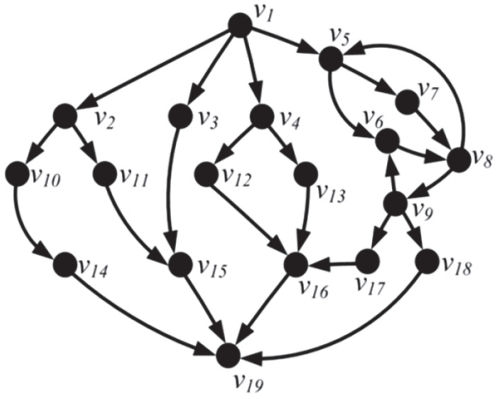
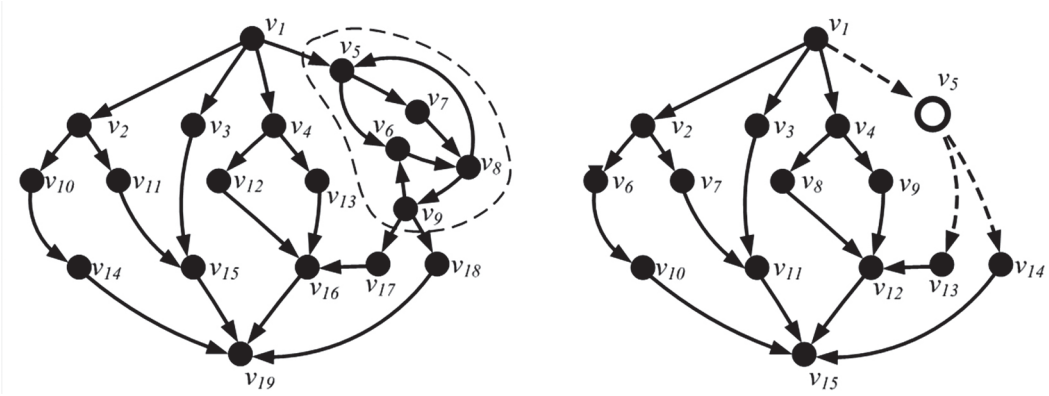


Рис. 1. Граф-модель ИС

Множеству вершин V сопоставляется множество макроопераций M, которые выполняются определенными программными модулями. Множеству дуг сопоставляется множество потоков данных D между модулями. При этом каждая дуга $x_{i,j} \in X$ характеризуется парой (d_{ij},t_{ij}), где $d_{i,j} \in D$ – пересылаемые данные между v_i и v_j вершинами, а t_{ij} – время, необходимое на пересылку этих данных. Каждый программный модуль $m_k \in M, \forall k = \overline{1,n}$, где n – общее количество модулей, в свою очередь, характеризуется парой (n_k,s_k), где n_k – имя модуля, s_k – его размер. Размер модуля определяется количеством элементарных операций, входящих в его состав. На орграфе задается входная вершина v_1, для которой $deg^+(x_0)=0$ и выходная вершина v_n, для которой $deg^-(x_n)=0$. При этом для указанных вершин выполняются условия, такие что каждая вершина $v_i \in V, \forall i = \overline{1,n}$ достижима из входа v_1 и каждая вершина $v_i \in V, \forall i = \overline{1,n}$ достигает выхода v_n. К орграфу (1) предъявляется требование ацикличности, отсутствия парных ребер и петель. Эти требования обеспечиваются заданием уровня детализации граф-модели. Формально это выглядит



а *б*

Рис. 2. Макропроектирование структуры ИС

следующим образом. На заданном графе (рис. 2, *а*) выделяются области с циклами μ $[v_b, v_e]$, где v_b – вершина начала цикла, v_e – его конец. Далее все вершины данного цикла исключаются из графа и заменяются одной вершиной v_p, для которой индекс p (рис. 2, *б*) определяется как минимальный номер вершины, которая входит в цикл . У всех остальных вершин индекс заменяется по правилу: если $ind > p$, тогда $Nind = ind - p + 1$, где ind – текущий индекс вершины, $Nind$ – новый индекс вершины, p – номер вершины, соответствующей исключенному из графа циклу. По этому принципу первоначальный граф, описывающий предметную область, преобразуется в граф конденсации, называемый графом Герца [3]. Могут возникать случаи, когда цикл не начинается вовсе или исполнение цикла прерывается по каким-либо причинам и переход по связи происходит в другую вершину, которая не принадлежит множеству вершин заданного цикла. В первом случае v_b будет совпадать с v_e. Во втором случае в процессе конденсации необходима дополнительная дуга, учитывающая возможность прерывания цикла. После того, как граф приведен к ациклической форме, на нем выполняется алгоритм приведения к ЯПФ (рис. 3):

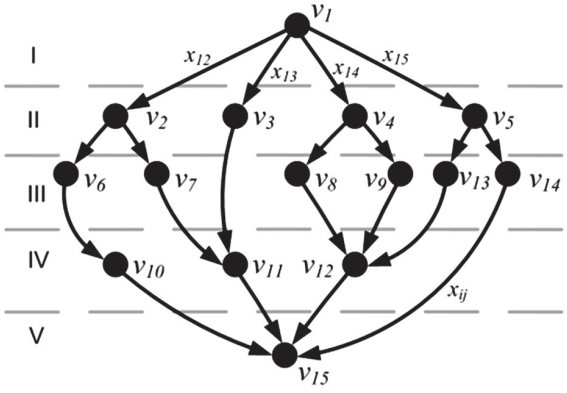


Рис. 3. Ярусно-параллельная форма графа

1) просматривается матрица смежности графа и в очередной ярус выбираются вершины с нулевой полустепенью захода, соответствующие нулевым столбцам;

2) строки, соответствующие выбранным на предыдущем шаге вершинам, обнуляются;

3) осуществляется возврат к шагу 1, до полного исчерпания матрицы смежности;

4) прорисовываются дуги.

Учитывая тот факт, что между модулями существуют связи, и один модуль может использовать переменные, константы и программный код другого модуля, при компиляции модулей в единую ИС, необходимо формально задать требования к структуре каждого программного модуля в виде множества спецификаций. Спецификацией программы будем называть набор требований к ней. Пусть P – программа на некотором языке программирования, а Sp – спецификация. Задача верификации программы P относительно спецификации Sp состоит в определении того, что P соответствует всем требованиям, содержащимся в Sp. Другими словами, для i-го программного модуля определяется спецификация $Sp_i \in SPC, \forall i = \overline{1,n}$, где SPC – множество спецификаций, вида:

$$Sp_i = \{D^{in}, D^{out}, T^{in}, T^{out}, R^{in}, R^{out}, F\}, \qquad (2)$$

где D^{in} – множество входных массивов данных; D^{out} – множество выходных массивов данных; T^{in} – множество типов входных данных; T^{out} – множество типов выходных данных; R^{in} – множество правил ввода входных данных модуля; R^{out} – множество правил вывода выходных данных модуля; F – множество управляющих воздействий i-го модуля.

Следующим этапом является анализ ПрО. Выделяются основные элементы и связи между ними. Такой анализ приводит к построению граф-модели ПрО вида

$$Pr = (Vpr, Xpr), \qquad (3)$$

где Vpr – множество вершин графа ПрО, которым соответствуют подсистемы объекта автоматизации (функциональная подсистема), а Xpr – множество ориентированных дуг, соединяющих вершины, которым соответствуют потоки данных между подсистемами. Так как топологии при заданных графах (1) и (3) должны соответствовать друг другу, выполняется функтор отображения вида:

$$\vartheta : Pr \to G . \quad (4)$$

Для обеспечения хранения информации о выполненных вычислениях, текущем состоянии ИС и предоставления этой информации по запросам как пользователя, так и некоторого программного модуля, используются кортежи данных.

Кортеж данных *C* для каждой вершины v_i, который в дальнейшем будет базой для организации хранения информации о работе ИС в базе данных, определяется следующим образом:

$$C_i = \{User\_ID, Time\_Op, Num\_V, Data\_Res, Data\_Inp, PCall\_Par\}, \quad (5)$$

где *User_ID* – идентификатор пользователя, активировавшего процедуру; *Time_Op* – время активации процедуры; *Num_V* – номер вершины, инициализирующей вычислительный процесс; *Data_Res* – ссылка на массив данных, полученных в результате вычислений *i*-ой процедуры; *Data_Inp* – ссылка на входные массивы данных для *i*-ой процедуры; *PCall_Par* – параметры вызова *i*-ой процедуры. Входные и выходные данные процедуры записываются в виде векторов данных:

$$Data\_Res = \{d_q^{RES}\}, \forall q = \overline{1,Q} \quad (6)$$

$$Data\_Inp = \{d_p^{INP}\}, \forall p = \overline{1,P} , \quad (7)$$

где *Q, P* – количество всех выходных и входных элементов данных. Сформированное формальное описание ИС и ПрО используется далее для построения детальных графовых моделей (таких как информационный граф или граф управления). В дальнейшем по полученным моделям производится построение и отладка программного средства.

2. Графовые модели

Выделяют четыре основные графовые модели: граф управления, информационный граф, операционно-логическая история и история реализации [2]. Первые две модели не зависят от входных данных и строятся непосредственно по тексту программы. Две последние модели для своего построения формально требуют слежения за выполнением всех операндов. Сложность построения модели возрастает в порядке указанного перечисления. Достоинством всех моделей является то, что они существуют для всех программ.

Если задать вектор исходных данных (7) и проследить за выполнением программы, зафиксировав каждое срабатывание оператора отдельной вершиной, можно получить ориентированный граф, называемый историей реализации программы. Объединяя такие графы на различных векторах входных данных (7), получаем информационный граф. Такой граф представляет взаимодействие данных и команд. Информационный граф описывает последовательность выполнения операций и взаимную зависимость между различными макрооперациями или блоками операций. Узлами информационного графа являются макрооперации, а однонаправленными дугами – каналы обмена данными [4]. Принцип организации вычислений в ИС подразумевает поиск зависимостей от данных других программных модулей для текущей, активной, процедуры, которая запущена пользователем. Для реализации данного принципа требуется наличие информационного графа (рис. 4, а), на котором выделяется подграф (рис. 4, б), позволяющий описать зависимость программных модулей.

Формально этот алгоритм записывается следующим образом. Пусть задан граф

$$G' = (V', X') , \quad (8)$$

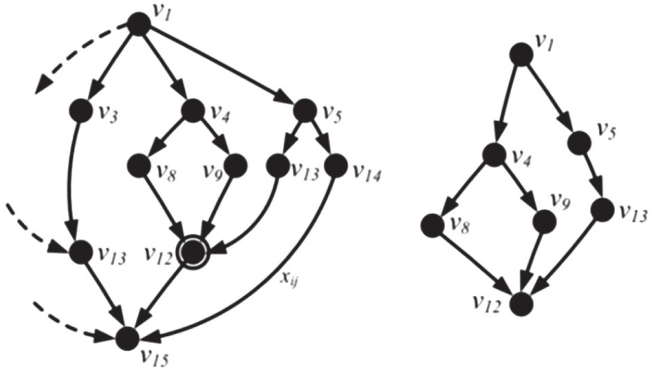
где $V' \subset V, X' \subset X$; V – множество вершин; X – множество дуг графа (1), а матрицы $A = \|a_{ij}\|$ и являются матрицами смежности для графа (1) и (8) соответственно. Тогда для выделения подграфа (8) на графе (1) необходимо:

1) запомнить индекс текущей вершины *k*, для которой определяются зависимости;

2) найти все элементы $a_{ij} \neq 0$ матрицы A при $i = k, \forall j = \overline{1,n}$, а затем переписать их в матрицу A' при соответствующих *i* и *j*;

3) запомнить индексы *j* для всех найденных вершин и, перебирая значения $k = j$ повторить шаг 2;

4) выполнять шаги пока не будет найден индекс для вершины со значением $deg^+(x_0)$=0.



а *б*

Рис. 4. Выделение подграфа для активной вершины

По окончании алгоритма будет сформирован подграф (8) с матрицей смежности , позволяющий в дальнейшем сформировать граф истории реализации программы (рис. 4, б).

Полученные графовые модели ИС позволяют определить тип модели параллельного программирования, используемой при разработке ИС.

3. Модели параллельного программирования

В зависимости от архитектур вычислительных систем и инструментальных средств, применяемых для разработки программных средств, выделяют различные модели параллельного программирования [2]:

Модель передачи сообщений. Программа порождает несколько задач, каждой задаче присваивается свой уникальный идентификатор, взаимодействие осуществляется посредством отправки и приема сообщений, новые задачи могут создаваться во время выполнения параллельной программы, несколько задач могут выполняться на одном процессоре.

Модель параллелизма данных. Одна операция применяется ко множеству элементов структуры данных, программа содержит последовательность таких операций, зернистость вычислений мала. Программист в этом случае указывает транслятору, как данные следует распределять между задачами.

Модель общей памяти. Задачи обращаются к общей памяти, имея общее адресное пространство и выполняя операции считывания/записи. Управление доступом к памяти осуществляется с помощью разных механизмов, таких, например, как семафоры. В рамках этой модели не требуется описывать обмен данными между задачами в явном виде, что упрощает программирование.

Основываясь на сформированных спецификациях программных модулей (2) и учитывая древовидную структуру ИС, можно сделать вывод, что наиболее оптимальной моделью программирования будет являться модель передачи сообщений, которая минимизирует нагрузку на каналы передачи данных, организовывая работу зависимых модулей ИС посредством только управляющих воздействий.

В качестве основного элемента, обеспечивающего управление ИС, берется так называемый семафор – специальный тип данных, принимающих значения из множества {0,1}, на котором определены две операции [5]. Эти операции обозначаются *P* и *V*. Если *s* – семафор, то *P(s)* имеет результатом *s-1*, но выполняется только если $s > 0$. *V(s)* имеет результатом *s+1*. Подобный элемент управляет состояниями ИС, в которые она переходит под воздействием вводимых данных. Поэтому такой процесс удобнее всего будет описать с использованием теории автоматов.

4. Конечный автомат в качестве модели ИС

Конечный автомат (КА) удобно использовать для структурирования приложения, организации и сопровождения логики пользовательского интерфейса. Он используется для управления набором ресурсов, которые должны удерживаться в памяти в любой момент времени. При этом состояние приложения определяется как совокупность значений всех его переменных. Значения меняются под воздействием внешнего события, а для определения состояния КА возможно использовать переменную перечисления, которая сохраняется в памяти. Каждый конкретный набор данных всегда будет переводить КА в одно и то же множество состояний, которое называется предысторией КА. В случае эквивалентных предысторий, если они одинаковым образом влияют на дальнейшее поведение автомата, нет необходимости запоминать все входные истории, а хранить в памяти их класс эквивалентности [5], что обеспечивает оптимизацию памяти ИС. Кроме того, КА позволяет осуществлять централизованное явное управление ПС [6]. Определяется КА следующим образом:

$$A = \{S, X, Y, s_0, \delta, \lambda\}, \qquad (10)$$

где S – конечное непустое множество состояний; X – конечное непустое множество входных сигналов (входной алфавит); Y – конечное непустое множество выходных сигналов (выходной алфавит); $s_0 \in S$ – начальное состояние; $\delta : S \times X \to S$ – функция переходов; $\lambda : S \times X \to Y$ – функция выходов. Определим КА информационной системы для заданной модели ПрО в виде графа переходов (рис. 5). Для такого графа задается вершина s_0 – начальное состояние ИС – ожидание ввода данных. В качестве данных выступает номер модуля, который должен быть запущен в заданный момент времени или для которого должен быть выполнен откат системы до заданного состояния. Режиму ожидания соответствует переход a_{00}. Переход a_{01} – активация вычислительного процесса; переход a_{05} – активация процесса отката системы. Вершина s_1 определяет подграф зависимостей между модулями ИС и текущим заданным модулем.

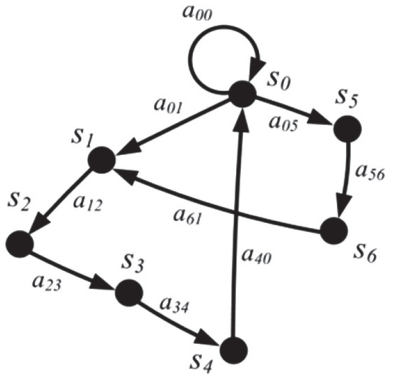


Рис. 5. КА состояний ИС
при заданной модели ПрО

Переход a_{12} – передача управления функции поиска при успешном выделении подграфа зависимостей. Вершине s_2 соответствует поиск сохраненной предыстории вычислений и восстановление из памяти массивов данных. Переход a_{23} – передача управления текущему модулю ИС. Вершине s_3 соответствует активация вычислительного процесса заданного модуля и дополнение предыстории вычислений с последующим сохранением (вершина s_4). Переход a_{40} – означает возвращение в режим ожидания. Вершина s_5 - ввод данных отката системы до заданного состояния, s_6 – выделение части

подграфа, необходимого для пересчета, то есть тех модулей, результаты работы которых будут обновляться. Переход a_{61} – активация функции выделения подграфа зависимостей при заданном подграфе, определяющем массивы данных для обновления.

Выводы

Формальное описание структуры ПС позволяет выделить такие основные элементы как 1) вершина граф-модели ИС, которая соответствует программному модулю и являет собой описание макропроцесса заданной ПрО; 2) ориентированная дуга граф-модели, связывающая на информационном уровне модули ПС и являющаяся каналом передачи данных; 3) спецификация модуля, которая задает стандарты взаимодействия между модулями ПС и компиляции модулей в единую структуру ИС; 4) кортеж данных, представляющий собой базис, необходимый для обеспечения хранения и восстановления информации о результатах работы программы; 5) подграф, позволяющий сохранять историю реализации ИС.

Сформированный КА информационной системы описывает ее макросостояния и позволяет выполнить централизованное управление ИС, что дает преимущество при разработке и проектировании ПС для заданных моделей ПрО.

Описанные и структурированные элементы граф-модели ПрО позволяют детально разработать алгоритмы функционирования исполнительной системы [1] и механизмы управления вычислительными процессами и обмена данными между ними, что является перспективой данного исследования. А также построить математическую модель, описывающую ИС как динамическую информационную систему, которая дает возможность оценить характеристики надежности ИС.

Список литературы: 1. *Чайников С.И.* Принципы организации вычислений на базе граф-модели предметной области [Текст] / С.И. Чайников, А.С. Солодовников // Бионика интеллекта: науч.-техн. журнал. – 2012. – №2 (79). – С. 72-75. **2.** *Воеводин В.В.* Параллельные вычисления [Текст] / В.В. Воеводин, Вл. В. Воеводин. – СПб.: БВХ-Петербург, 2002. – 608 с. **3.** *Евстигнеев В.А.* Применение теории графов в программировании [Текст] / В.А. Евстигнеев. – М.: Наука, 1985. – 352 с. **4.** *Немнюгин С.А.* Модели и средства программирования для многопроцессорных вычислительных систем [Текст] / С.А. Немнюгин. – СПб. : С.-Петербургский ГУ, 2010. – 100 с. **5.** *Карпов Ю. Г.* Теория автоматов [Текст] / Ю. Г. Карпов. – СПб.: Питер, 2003. – 208 с. **6.** *Салмре И.* Программирование мобильных устройств на платформе .NETCompactFramework / Иво Салмре, пер. с англ. – М.: Изд. дом «Вильямс», 2006. – 736 с.

Поступила в редколлегию 20.12.2012

УДК 004(4’242+053)

Формалізований опис ГРАФ-моделі предметної галузі / С.І. Чайніков, А.С. Солодовніков // Біоніка інтелекту: наук.-техн. журнал. – 2013. – № 1 (80). – С. 77-81.

Пропонується формалізований опис елементів діалогової системи та предметної галузі. Розглянуто механізми організації даних, необхідних для виконання обчислень та відновлення збереженої інформації у діалоговій системі. Запропоновані механізми конденсації граф-моделі предметної галузі та укрупнений опис стану інформаційної системи на базі теорії автоматів.

Іл. 5. Бібліогр.: 6. найм.

UDC 004(4’242+053)

The formalized description of problem domain graph-model / S.I. Chaynikov, A.S. Solodovnikov // Bionics of Intelligense: Sci. Mag. – 2013. – № 1 (80). – P. 77-81.

The formalized description of main elements of a dialog system and problem domain is proposed. The mechanisms of data organization that computing processes need at and problems of data storing and restoration are considered. The principles of graph-model condensation and aggregated description of iformation system states based on automata theory are proposed.

Fig. 5. Ref.: 6 items.