УДК 519.876.5

Иерархический метод анализа функционирования программного обеспечения на основе сети Петри

© И.В. Рудаков, А.В. Пащенкова

МГТУ им. Н.Э. Баумана, Москва, 105005, Россия

Статья посвящена иерархическому методу анализа функционирования программного обеспечения, позволяющему обнаружить ошибки проектирования на ранних этапах разработки. Рассмотрен блочно-иерархический подход к проектированию. Приведены иерархические структуры программных систем, использование которых позволяет сделать сложные программные системы обозримыми. Изложенный метод основывается на представлении программного обеспечения иерархической сетью Петри. Обнаружение ошибок проектирования происходит посредством анализа полученной сети Петри.

***Ключевые слова*:** *блочно-иерархический подход, декомпозиция, иерархические структуры, сеть Петри.*

Введение. Сложные программные системы характеризуются большим разнообразием взаимосвязей элементов, обработкой больших массивов информации, элементов конкуренции при использовании ресурсов ЭВМ. Известно, что разработка программного обеспечения — сложный многоэтапный процесс, включающий в себя этапы анализа, непосредственного написания, тестирования и внедрения [4]. Значительное упрощение понимания сложных задач в процессе разработки достигается за счет образования иерархической структуры из абстракций и (или) модулей. При проектировании программных систем возникает необходимость в исследовании взаимодействия элементов системы.

Одним из известных методов исследования процесса функционирования сложных систем является их формализация сетью Петри [1]. Данный математический аппарат позволяет формировать адекватные модели сложных систем с иерархической структурой и разрабатывать оптимальные алгоритмы решения задач.

Блочно-иерархический подход к проектированию программного обеспечения. Известно, что в процессе проектирования программного обеспечения формируются определенные представления о системе, отражающие ее существенные свойства с той или иной степенью подробности. В этих представлениях выделяют составные части — уровни проектирования, которые, в свою очередь, подразделяют на горизонтальные (иерархические) и вертикальные уровни проектирования (их называют также аспектами проектиро-

вания). Представления о сложных объектах внутри каждого аспекта необходимо разделять на иерархические уровни (уровни абстрагирования).

Выделение горизонтальных уровней лежит в основе блочно-иерархического подхода к проектированию [4]. На верхнем уровне используют наименее детализированное представление, отражающее только самые общие черты и особенности проектируемой системы. На следующих уровнях степень подробности описания возрастает, при этом рассматривают уже отдельные блоки системы, но с учетом воздействий на каждый из них его соседей. Такой подход позволяет на каждом иерархическом уровне формулировать задачи приемлемой сложности, поддающиеся решению с помощью имеющихся средств проектирования. При этом описание каждого блока не должно быть слишком подробным, так как это приведет к чрезмерной громоздкости описаний и невозможности решения возникающих проектных задач. Разбиение на уровни должно быть таким, чтобы документация на блок любого уровня была обозрима и воспринимаема одним человеком [4].

Проектирование программного обеспечения, как и любых других сложных систем, выполняется поэтапно с использованием блочно-иерархического подхода, который, как было описано ранее, основан на разбиении сложной задачи большой размерности на последовательно и (или) параллельно решаемые группы задач малой размерности. Такой подход позволяет разбивать исследуемый объект на компоненты требуемой степени детализации и проверять правильность функционирования каждой из компонент.

Декомпозиция сложных систем. Блочно-иерархический подход к проектированию основан на декомпозиции сложных объектов и, соответственно, средств их создания на иерархические уровни и аспекты. При декомпозиции учитывают, что связи между отдельными частями должны быть слабее, чем связи элементов внутри частей. Кроме того, чтобы из полученных частей можно было собрать разрабатываемый объект, в процессе декомпозиции необходимо определить все виды связей частей между собой [4]. При создании сложных объектов процесс декомпозиции выполняется многократно: каждый блок, в свою очередь, декомпозируют на части, пока не получат блоки, которые сравнительно легко разработать. Данный метод разработки получил название пошаговой детализации.

Результат декомпозиции обычно представляют в виде схемы иерархии, на нижнем уровне которой располагают сравнительно простые блоки, а на верхнем — объект, подлежащий разработке (рис. 1). На каждом иерархическом уровне описание блоков выполняют с определенной степенью детализации, абстрагируясь от несущественных деталей. Следовательно, для каждого уровня используют свои

формы документации и свои модели, отражающие сущность процессов, выполняемых каждым блоком. Так, для объекта в целом, как правило, удается сформулировать лишь самые общие требования, а блоки нижнего уровня должны быть специфицированы так, чтобы из них действительно можно было собрать работающий объект. Другими словами, чем больше блок, тем более абстрактным должно быть его описание.

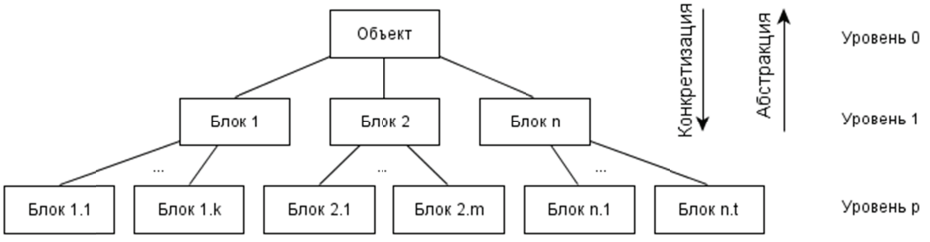

Рис. 1. Соотношение абстрактного и конкретного в описании блоков при блочно-иерархическом подходе

Помимо того, что использование блочно-иерархического подхода делает возможным создание сложных систем, он также:

– упрощает проверку работоспособности как системы в целом, так и отдельных блоков;

– обеспечивает возможность модернизации систем, например, замены ненадежных блоков с сохранением их интерфейсов.

Существенно, что в процессе декомпозиции следует выделять аналогичные блоки, которые можно было бы разрабатывать на общей основе. Таким образом обеспечивают увеличение степени повторяемости кодов и, соответственно, снижение стоимости разработки.

При соблюдении принципа пошаговой детализации разработчик сохраняет возможность осмысления проекта и, следовательно, может принимать наиболее правильные решения на каждом этапе, что называют локальной оптимизацией (в отличие от глобальной оптимизации характеристик объектов, которая для действительно сложных объектов не всегда возможна).

При проектировании программного обеспечения после определения его общей структуры выполняют декомпозицию компонентов в соответствии с выбранным подходом до получения элементов, которые, по мнению проектировщика, в дальнейшей декомпозиции не нуждаются. Среди способов декомпозиции разрабатываемого программного обеспечения можно выделить функционально-модульный (или структурный) и объектно-ориентированный [4].

Результатом структурной декомпозиции является иерархия подпрограмм (процедур), в которой функции реализуются подпрограм-

мами верхних уровней, а непосредственно обработка — нижних уровней. Любая подпрограмма возвращает управление той подпрограмме, которая ее вызвала.

Результатом объектной декомпозиции является совокупность объектов, которые затем реализуют как переменные некоторых специально разрабатываемых типов (классов), представляющих собой совокупность полей данных и методов, работающих с этими полями. Таким образом, при любом способе декомпозиции получают набор связанных с соответствующими данными подпрограмм, которые в процессе реализации организуют в модули.

Основными видами иерархических структур применительно к сложным программным системам являются структура классов (иерархия «целое-часть») и структура объектов (иерархия «простое-сложное»).

Одним из примеров иерархии является одиночное наследование. Важным элементом объектно-ориентированных систем и основным видом иерархии «целое-часть» является концепция наследования. Наследование означает такое отношение между классами (отношение родитель/потомок), когда один класс заимствует структурную или функциональную часть одного или нескольких других классов (соответственно, одиночное и множественное наследование). Иными словами, наследование создает такую иерархию абстракций, в которой подклассы наследуют строение от одного или нескольких суперклассов. Часто подкласс достраивает или переписывает компоненты вышестоящего класса.

Иерархическая структура программной системы — основной результат предварительного проектирования. Она определяет состав модулей программной системы (иерархию процедур и функций) и управляющие отношения между модулями. При этом сама программа является первым уровнем в данной иерархии. Модульная структура отражает только тот факт, что модуль более высокого уровня иерархии вызывает все непосредственно связанные с ним модули более низкого уровня иерархии, не показывая при этом порядок вызова связанных модулей более низкого уровня. Иерархическая структура не отражает процедурные особенности программной системы, т. е. последовательность операций, их повторение, ветвления и т. д. На рис. 2 представлены основные характеристики иерархической структуры.

Первичными характеристиками являются количество вершин (модулей) и количество ребер (связей между модулями). К ним добавляются две глобальные характеристики — высота (количество уровней управления) и ширина (максимальное из количеств модулей, размещенных на уровнях управления).

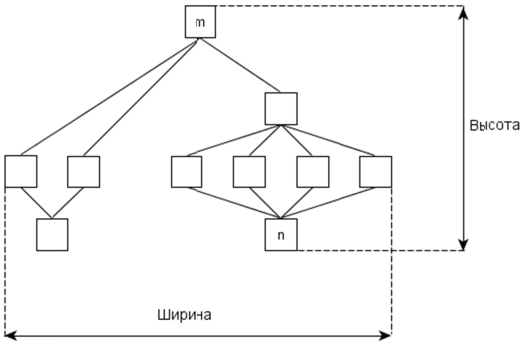

Рис. 2. Иерархическая структура программной системы

Метод формализации программного обеспечения сетями Петри. На рис. 3 показана функциональная диаграмма метода исследования программного обеспечения, формализованного иерархической сетью Петри.

Рис. 3. Функциональная диаграмма метода анализа программного обеспечения, формализованного сетью Петри

Процесс проектирования сложных программных систем неразрывно связан с UML (Unified Modeling Language — унифицированный язык моделирования) — стандартом, позволяющим разработчикам формулировать мысли и общаться между собой. Однако UML — всего лишь система обозначений, основанная на диаграммах, она не может защитить от ошибок проектирования. Скрупулезный анализ и хорошо спланированное тестирование позволяют исключить ошибки, которые, тем не менее, в достаточно сложных системах могут быть выявлены уже в процессе эксплуатации. Стоимость исправления таких ошибок может быть весьма существенной. Моделирование проектируемой системы сетью Петри благодаря развитым инструментам позволяет разработчикам эффективно обнаруживать и исправлять ошибки на ранних стадиях [2].

Для преобразования диаграмм в сеть Петри необходимо выполнить пять шагов [3].

Шаг 1. С использованием результатов этапа анализа создается диаграмма классов (class diagram).

Шаг 2. Для каждого класса системы создается диаграмма состояний, отражающая последовательности состояний, в которые попадает класс при его создании, удалении и вызове его методов (statechart diagram).

Шаг 3. Взаимодействие классов фиксируется на диаграмме последовательности (sequence diagram).

Шаг 4. Каждая диаграмма состояний с помощью набора правил преобразуется в страницу раскрашенной иерархической сети Петри. Для преобразования диаграмм состояний и активностей в сеть Петри необходимо обозначить правила перевода (рис. 4).

Шаг 5. Все полученные страницы иерархической сети Петри связываются в единую сеть на основании диаграммы последовательности.

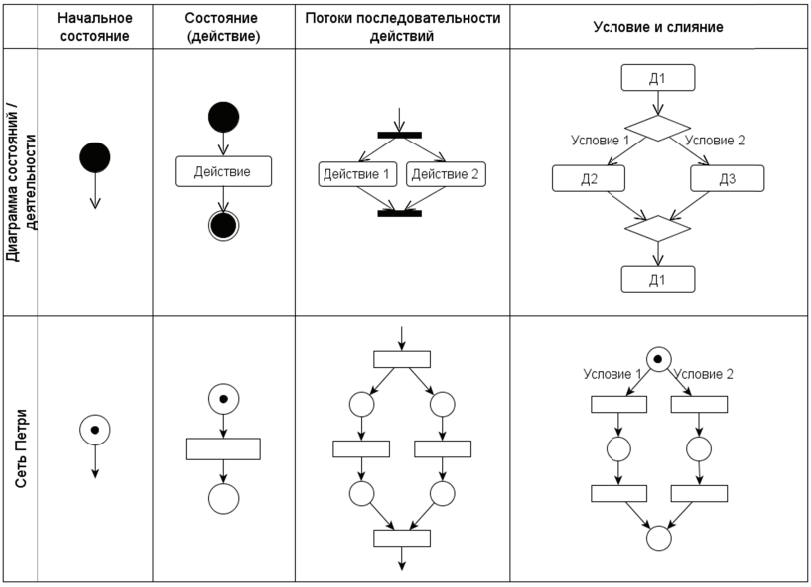

Рис. 4. Соответствие элементов UML-диаграмм конструкциям сети Петри

На основании проведения анализа полученной сети Петри делается вывод о правильности функционирования модели, а в случае обнаружения ошибок и неточностей производится корректировка сети Петри и UML-диаграмм.

Одним из методов анализа сети Петри является дерево достижимости [1], являющее собой множество достижимости сети. Корневая вершина представляет первоначальную маркировку. Из каждой вершины исходят дуги, соответствующие разрешенным переходам. Всякий путь в дереве, начинающийся в корне, соответствует допустимой

последовательности переходов. Сеть Петри может иметь бесконечное дерево достижимости. Для получения дерева, которое можно считать полезным инструментом анализа, необходимо найти средства ограничения его до конечного размера.

Особенностью алгоритма построения конечного дерева достижимости является специальная классификация маркировок. Каждая вершина дерева классифицируется как граничная, терминальная, дублирующая или внутренняя вершина. Граничными являются вершины, которые еще не обработаны алгоритмом. После обработки граничные вершины становятся либо терминальными, либо дублирующими, либо внутренними. Маркировки, в которых нет разрешенных переходов, являются терминальными вершинами дерева достижимости. Другой класс маркировок — это маркировки, ранее встречавшиеся в дереве. Они называются дублирующими вершинами. Никакие последующие маркировки рассматривать не нужно, так как все они будут порождены из места первого появления дублирующей маркировки в дереве.

Для сведения дерева достижимости к конечному представлению используется еще одно средство. Для позиций, которые увеличивают число фишек некоторой последовательностью запусков переходов, можно создать произвольно большое число фишек, просто повторяя данную последовательность столько раз, сколько это нужно. Бесконечное число маркировок, получающихся из циклов такого типа, представляется с помощью специального символа w, который обозначает «бесконечность». Таким образом, в маркировке число фишек может быть неотрицательным целым либо w.

Первый шаг алгоритма определяет начальную маркировку корнем дерева, т.е. граничной вершиной. Последующие шаги направлены на обработку граничных вершин. До тех пор, пока имеются граничные вершины, они обрабатываются алгоритмом.

Пусть x — граничная вершина, которую необходимо обработать.

1. Если в дереве имеется другая вершина y, не являющаяся граничной, и с ней связана та же маркировка ($\mu[x] = \mu[y]$), то вершина x — дублирующая.

2. Если для маркировки $\mu[x]$ ни один из переходов не разрешен (т. е. $\delta(\mu[x], t_j]$) не определено для всех $t_j \in T$, $\delta(\mu[x], t_j])$ — функция следующего состояния), то x — терминальная вершина.

3. Для всякого перехода $t_j \in T$, разрешенного в $\mu[x]$ (т.е. $\delta(\mu[x], t_j)$ определено), создать новую вершину z дерева достижимости. Маркировка $\mu[z]$, связанная с этой вершиной, определяется для каждой позиции p_i следующим образом:

– если $\mu[x]_i = w$, то $\mu[z]_i = w$;

– если на пути от корневой вершины к x существует вершина y с $\mu[y] < \delta(\mu[x], t_j)$ и $\mu[y]_i < \delta(\mu[x], t_j)_i$, то $\mu[z]_i = w$;

– в противном случае $\mu[z]_i = \delta(\mu[x], t_j)_i$.

Дуга, помеченная t_j, направлена от вершины x к вершине z. Вершина x переопределяется как внутренняя, z — становится граничной. Когда все вершины дерева — терминальные, дублирующие или внутренние, алгоритм останавливается. На рис. 5 показана блок-схема алгоритма построения конечного дерева достижимости.

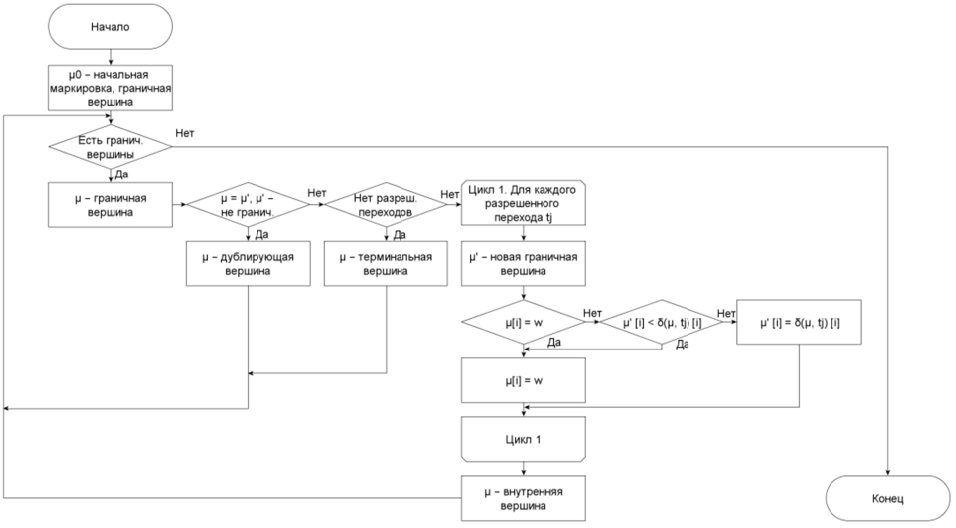

Рис. 5. Алгоритм построения конечного дерева достижимости

Особенностью данного метода анализа алгоритма программного обеспечения, формализованного иерархической сетью Петри, является необходимость различать успешное завершение работы подсети от тупиковой ситуации. Для этого предлагается вводить дополнительно конечное состояние сети [1].

Другой подход к анализу сетей Петри основан на их матричном представлении и решении матричных уравнений. Матричный подход основывается на представлении сети двумя матрицами D^- и D^+, представляющими входную и выходную функции сети. Каждая матрица имеет m строк (по одной на переход) и n столбцов (по одному на позицию). D^- описывает входы в переходы, D^+ — выходы из переходов. $D = D^+ - D^-$ — составная матрица изменений.

Матричная теория является инструментом для решения проблемы достижимости. Пусть маркировка M' достижима из M_0, тогда существует последовательность (возможно пустая) запусков переходов G, которая приводит из M_0 к M'. Это означает, что $f(G)$ является неотрицательным целым решением следующего матричного уравнения для x:

$$M' = M_0 + xD\,. \qquad (1)$$

Следовательно, если M' достижима из M_0, тогда уравнение (1) имеет решение в неотрицательных целых. Если уравнение не имеет решения, тогда M' недостижима из M_0.

Однако в матричном методе анализа существует ряд недостатков:

– матрица D теряет информацию о ситуациях, когда переходы имеют входы и выходы из одной позиции (петли);

– отсутствие информации о последовательности в векторе запуска. Хотя известно число переходов, порядок их запуска неизвестен;

– решение уравнения (1) является необходимым для достижимости, но недостаточным.

Матричный метод анализа функционирования программного обеспечения, формализованного иерархической сетью Петри, позволяет найти недостижимые участки, т.е. те части программы, которые никогда не выполняются [1].

Заключение. В процессе проектирования программного обеспечения использование блочно-иерархического подхода значительно упрощает понимание решения сложных задач. При этом результатом декомпозиции программного обеспечения является набор связанных с соответствующими данными процедур, которые в процессе реализации организуются в модули. Вследствие возможности ошибок проектирования возникает необходимость в исследовании взаимодействия элементов программной системы. Предложенный метод анализа программного обеспечения, формализованного иерархической сетью Петри, позволяет получить информацию о наличии взаимоблокировок, невыполнимых операций, зацикливаний, что повышает надежность разрабатываемого программного обеспечения. Метод реализован в виде законченного программного комплекса и протестирован на ряде известных алгоритмов, подтвердив свою работоспособность.

ЛИТЕРАТУРА

[1] Рудаков И.В., Пащенкова А.В. Программный комплекс верификации алгоритмов программного обеспечения с помощью иерархических сетей Петри. *Вестник МГТУ им. Н.Э. Баумана: электронное издание*, 2013, № 2(14), с. 10.

[2] Воевода А.А., Прытков Д.В. Применение сетей Петри на этапе объектно-ориентированного проектирования. *Сб. научных трудов НГТУ*, 2012, № 2(60), с. 65–76.

[3] Коротиков С.В. *Применение сетей Петри в разработке программного обеспечения центров дистанционного контроля и управления*. Дисс. … канд. техн. наук. Новосибирск, НГТУ, 2007, 216 с.

[4] Норенков И.П. *Основы автоматизированного проектирования*. Москва, Изд-во МГТУ им. Н.Э. Баумана, 2002, 336 с.

Статья поступила в редакцию 10.06.2013

Ссылку на эту статью просим оформлять следующим образом:

Рудаков И.В., Пащенкова А.В. Иерархический метод анализа функционирования программного обеспечения на основе сети Петри. *Инженерный журнал: наука и инновации*, 2013, вып. 6. URL: http://engjournal.ru/catalog/it/hidden/779.html

Рудаков Игорь Владимирович — канд. техн. наук, заведующий кафедрой «Программное обеспечение ЭВМ и информационные технологии» МГТУ им. Н.Э. Баумана. e-mail: irudakov@yandex.ru

Пащенкова Анна Валерьевна — студентка кафедры «Программное обеспечение ЭВМ и информационные технологии» МГТУ им. Н.Э. Баумана. e-mail: anna_vp@inbox.ru