Моделирование наследования при построении модели объектно-ориентированной программы в терминах сетей Петри

Харитонов Д. И.[1], Тарасов Г. В.[2], Парахин Р. В.[3], Голенков Е. А.[4], Леонтьев Д. В.[5]

[1]Харитонов Дмитрий Иванович / Kharitonov Dmitriy Ivanovich – кандидат технических наук, старший научный сотрудник;
[2]Тарасов Георгий Витальевич / Tarasov Georgiy Vitalevich – научный сотрудник;
[3]Парахин Роман Валерьевич / Parakhin Roman Valeryevich – инженер-программист;
[4]Голенков Евгений Александрович / Golenkov Evgeniy Aleksandrovich – кандидат физико-математических наук, старший научный сотрудник;
[5]Леонтьев Денис Васильевич / Leontyev Denis Vasilevich – аспирант, инженер-программист,
Институт автоматики и процессов управления, г. Владивосток

Аннотация: *в статье представлен подход к построению моделей объектно-ориентированных программ, использующих инкапсуляцию и наследование, в терминах структурированных сетей Петри. Описан набор операций композиции структурированных сетей Петри, а также правила построения моделей для деклараций и реализаций основных компонент программы.*

Ключевые слова: *моделирование программ, проверка корректности программ на моделях, объектно-ориентированные программы, сети Петри.*

Введение

Теория сетей Петри широко применяется при моделировании и анализе программного обеспечения. Наиболее заметные работы в этой области связаны с исследованием свойств параллельных программ. Поиск тупиковых состояний [1, 2], анализ производительности программ [3, 4], проверка корректности обмена сообщениями [5] являются лишь некоторыми примерами применения сетей Петри. Необходимо отметить, что в этих и многих других работах особую важность имеют модели программ, обеспечивающие формальную основу для анализа программы целиком или некоторых характеристик её компонент. Формирование соответствующей модели исследуемой программ является одним из наиболее трудоёмких этапов в процессе моделирования, и определённая часть исследований направлена на автоматизацию создания моделей программ [6, 7, 8]. Принимая во внимание постоянное развитие языков программирования и сред разработки программ, расширение синтаксиса и семантики современных языков программирования, задача автоматического построения моделей программ из их исходного текста приобретает всё большую практическую значимость для анализа свойств программ и проверки их корректности.

Структурированные сети Петри

Для автоматизации моделирования объектно-ориентированных программ авторами предложено расширение сетей Петри - структурированные сети Петри. Строительным элементом структурированных сетей Петри является PNS-объект, состоящий из сети Петри, входящего дерева имён, исходящего дерева имён и множества входящих и исходящих точек доступа, образующих входящий и исходящий интерфейс объекта. Основная идея структурированных сетей Петри состоит в том, что в операциях композиции PNS-объектов выбор точек доступа для композиции будет выполняться автоматически. Для этого входящие точки доступа состоят из функции пометки переходов сети Петри и вершины из входящего дерева имён в качестве структурного указателя. Исходящие точки доступа также содержат функцию пометки переходов сети Петри, и вершины из исходящего дерева имён в качестве структурного указателя для автоматического определения точки доступа при композиции PNS-объектов. При этом именованный путь к корневой вершине исходящей точки доступа в одном PNS-объекте должен совпадать с именованным путём к корневой вершине входящей точки доступа в дереве имён другого PNS-объекта. Предваряя построение операций композиции PNS-объектов, были определены операции с точками доступа объекта:

• Операция объединения точек доступа позволяет слить в PNS-объекте две и более точки доступа, имеющие одинаковый структурный указатель. При этом функция пометки в результирующей точке доступа строится как сумма пометок исходных точек доступа. Записывается в виде: $E = (E')_{\gamma=\alpha+\beta}$.

• Операция ограничения по точке доступа удаляет из PNS-объекта все переходы, имеющие ненулевую пометку в заданной точке доступа и саму точку доступа. Записывается в виде: $E = \partial_{\alpha}(E')$.

• Операция ограничения по ветке точек доступа используется только для точек доступа входящего интерфейса, при этом выполняется ограничение по всем точкам доступа, имеющим структурный указатель, принадлежащий выбранной ветке точек доступа во входящем дереве имён PNS-объекта. Записывается в виде $E = \partial_{\bullet W}(E')$.

• Нормализация PNS-объекта выполняет объединение точек доступа с одинаковыми структурными указателями, а также удаляет все изолированные, т.е. не имеющие входящих или исходящих переходов, места в сети Петри PNS-объекта. Записывается в виде: $E = norm(E_1)$.

Далее в статье во всех операциях композиции используются предварительно нормализованные объекты. Для моделирования объектно-ориентированных программ, использующих инкапсуляцию и наследование, были разработаны следующие операции композиции PNS-объектов:

• Операция формального объединения строит из двух исходных один результирующий PNS-объект $E = E_1 \oplus E_2$, в котором сеть Петри является формальным объединением сетей Петри исходных объектов, входящее и исходящее дерево имён является формальным объединением соответствующих деревьев исходных объектов, а входной и выходной интерфейсы объекта являются объединением точек доступа соответствующих интерфейсов исходных объектов. Из определения операции следует, что она является коммутативной и ассоциативной, что позволяет формально объединять множество PNS-объектов без учёта порядка выбора пар объектов для выполнения операции.

• Операция простой композиции определяется для PNS-объекта и двух его точек доступа, по одной из входящего и исходящего интерфейса, имеющих совпадающие последовательности имён. В результате операции строится новый PNS-объект, сеть Петри которого отличается от исходной сети добавлением множества переходов синхронизации, каждый из которых имеет два прообраза в виде мультимножества переходов в исходной сети, пометки которых в точках доступа операции совпадают. Мультимножества входящих и исходящих мест для добавленных переходов совпадают с суммой мультимножеств соответствующих мест прообразов этих переходов. Пометка добавленных переходов по всем точкам доступа, кроме участвующих в операции, совпадает с суммой пометок их прообразов. В точках доступа, участвующих в операции, пометка новых переходов является нулевой. Бинарная версия операции сначала выполняет формальное объединение PNS-объектов, а затем унарную композицию:

$$E = [E_1]_{\beta}^{\alpha}\,,\ E = E_1\ {}_{\alpha}[\,]_{\beta} E_2 \equiv [E_1 \oplus E_2]_{\beta}^{\alpha}\,.$$

• Операция поглощения для нескольких PNS-объектов, $E = \boxplus(E', E_1, \ldots, E_n) \equiv \boxplus(E', (E_1 \oplus \cdots \oplus E_n)) \equiv \boxplus(q, X)$ выбирает множество пар точек доступа с одинаковыми последовательностями имён, в каждой из которых одна точка из входного интерфейса, а вторая точка доступа из выходного интерфейса, и выполняет операцию простой композиции для каждой пары. Затем по использованным точкам доступа из выходного интерфейса выполняется операция ограничения по ветке точек доступа. Далее для всех поглощаемых объектов (т.е. кроме выделенного) выполняется операция ограничения по входящим точкам доступа, так что далее любая синхронизация с этими объектами возможна только через выходной интерфейс.

• Операция структурной композиции (используется для моделирования инкапсуляции) для единственного PNS-объекта выбирает множество пар точек доступа с одинаковыми последовательностями имён, в каждой из которых одна точка доступа из входного интерфейса, а вторая точка доступа из выходного интерфейса. Для этого множества пар точек доступа выполняется операция простой композиции, а затем по использованным точкам доступа из выходного интерфейса выполняется операция ограничения по ветке точек доступа. Затем во входящее дерево имён добавляется новая корневая вершина, и достраиваются альтернативные пути к первому уровню вершин исходного дерева. В случае структурной композиции нескольких PNS-объектов сначала выполняется их формальное объединение, а затем унарная форма операции структурной композиции:

$$E = \curlywedge(q, E_1, \ldots, E_n) \equiv \curlywedge(q, (E_1 \oplus \cdots \oplus E_n)) \equiv \curlywedge(q, X).$$

• Операция иерархической композиции (используется для моделирования наследования) определяется для двух PNS-объектов $E = E_1 \circledcirc E_2$, сети Петри которых объединяются формально, исходящие деревья имён также объединяются формально, множества входящих и исходящих точек доступа объединяются, а входящие деревья имён объединяются с эксклюзивным порядком имён.

Для моделирования объектно-ориентированных программ, в частности программ на языке программирования C++, разработан следующий набор правил:

1. Модель декларации метода класса или глобальной функции f обозначается $Dcl(f)$ и строится однозначным способом в виде последовательности переход-место-переход, соединённых дугами и помеченных таким образом, что первый переход обозначает начало выполнения функции, а последний - конец выполнения функции.

2. Модель декларации класса обозначается $Dcl(C)$ и строится в две стадии. На первой стадии строится структурная композиция деклараций всех методов и членов, объявленных в этом классе:

$$E' = \curlywedge(C, Dcl(Meth) \oplus Dcl(Memb)),$$

На второй стадии выполняется иерархическая композиция полученной декларации с декларацией предка $E = E' \circledcirc E_{anc}$ или в случае множественного наследования - с декларациями всех классов предков по порядку наследования:

$$E = E' \circledcirc E_{a1} \circledcirc \ldots \circledcirc E_{an}\,.$$

3. Модель декларации объекта получается простым копированием PNS-объекта модели декларации класса с заменой входящего дерева имен на результат одношагового роста исходного дерева, параметризованного именем объекта.

4. Модель реализации метода класса или глобальной функции f получается в результате поглощения моделей декларации всех локальных переменных моделью потока управления функции или метода класса: $E \equiv Imp(f) = \boxplus\left(E_{cf}, Dcl(Var)\right)$. Построение модели потока управления может быть выполнено автоматически на основе синтаксически управляемой трансляции [9, 10].

5. Модель реализации класса получается структурной композицией моделей реализации всех его методов и моделей декларации всех членов этого класса, в число которых не включаются методы и члены, объявленные в классах предках:

$$E \equiv Imp(C) =\rightarrow (C, Imp(Meth) \oplus Dcl(Memb))$$

6. Модель программного модуля получается простым формальным объединением моделей реализации классов и функций, а также моделей декларации переменных, составляющих программный модуль:

$$E \equiv Imp(Q) = Imp(F) \oplus Imp(C) \oplus Dcl(V)$$

В C++ в качестве программного модуля можно рассматривать любой из файлов с исходными текстами программы или библиотеку функций, получаемую объединением многих файлов. Подобно C++, в результате формального объединения нескольких моделей программных модулей получается новый программный модуль.

7. Модель программы получается в результате операции поглощения моделью загрузчика программы программного модуля, содержащего все функции, переменные и классы программы: $E \equiv \boxplus (Loader, Imp(Q))$.

При этом модель программы будет считаться полной, если входной интерфейс полученного PNS-объекта будет пустым. На практике такая модель зачастую не может быть получена из исходных текстов программы, так как программа может обращаться к функциям библиотек, библиотеки обращаться к функциям операционной системы, операционная система - к драйверам и внутреннему программному обеспечению устройств вычислительной системы. Таким образом, можно либо удовлетвориться неполной моделью программы, подразумевая, что все неудовлетворённые вызовы обращаются к заведомо корректным функциям, либо выбрать набор библиотек и построить с участием человека их полностью автономные модели.

Процесс моделирования.

Представим кратко алгоритм построения модели объектно-ориентированной программы в терминах структурированных сетей Петри с использованием описанного выше аппарата структурированных сетей Петри. Для этого необходим транслятор с объектно-ориентированного языка программирования в некоторое древовидное представление, далее называемое деревом разбора. Каждый узел в этом дереве содержит имя, тип и роль.

Дерево разбора программы в целом строится слиянием деревьев разбора всех единиц трансляции программы (в нашем случае файлов). Слияние выполняется по именам и ролям узлов деревьев. При этом все повторные появления деклараций одних и тех же классов, объектов и функций объединяются.

Существует ряд ограничений, которым должна удовлетворять рассматриваемая программа. Тип узла в дереве разбора программы может быть: глобальным пространством, классом, методом, функцией, переменной и членом класса. Роль узлов может быть либо декларацией (declaration) либо реализацией (implementation). Также в программе не должно быть статических членов класса, а также не должен использоваться полиморфизм. Программа должна успешно проходить компиляцию, то есть не содержать синтаксических ошибок и проблем с неразрешимыми именами.

Алгоритм построения модели состоит из трёх стадий. На первой стадии рекурсивная процедура генерирует PNS-объекты для всех деклараций в программе, а также для реализаций всех функций и методов. Поскольку реализация классов разделена на множество реализаций его методов и членов, на первой стадии также подготавливается содержимое для объектов реализации классов, которые формируются на следующей стадии. И наконец, на третьей стадии все сгенерированные PNS-объекты объединяются вместе в модель программы с добавлением специального объекта, моделирующего первоначальную загрузку программы.

Заключение.

Автоматическое построение моделей объектно-ориентированных программ открывает целый ряд технических вопросов требующих формального решения.

Отличие объектно-ориентированных программ от, скажем, императивных в том, что единичные декларативные объявления в программе имеют влияние на то, как будет выполняться императивный текст многих функций и методов программы. В настоящей статье авторы предложили подход к моделированию инкапсуляции и наследования — двух из трёх концепций объектно-ориентированного программирования - в терминах структурированных сетей Петри. Для того чтобы сделать следующий шаг в моделировании ОО программ необходимо формализовать понятия указателей, динамического заведения объектов и полиморфизма. Ещё впереди предстоит моделирование статических функций и переменных, параметров функций, исключительных ситуаций, первоначального загрузчика программы, а также такого сложного понятия как шаблоны функций и классов.

Работа выполнена при поддержке программы фундаментальных исследований Президиума РАН «Фундаментальные проблемы математического моделирования» (проект 0262-2014-0157) и государственного финансирования (проект 0262-2014-0003).

Литература

1. *Lafortune S., Wang Y. and Reveliotis S*. Eliminating concurrency bugs in multithreaded software: An approach based on control of petri nets // Application and Theory of Petri Nets and Concurrency: 34th International

Conference. PETRI NETS 2013. Milan. Italy. June 24-28, 2013. Proceedings (J.-M. Colom and J. Desel eds.). Berlin. Heidelberg: Springer Berlin Heidelberg, 2013. P. 21–28.
2. *Wang Y., Kelly T., Kudlur M., Lafortune S. and Mahlke S.* Gadara: Dynamic deadlock avoidance for multithreaded programs // Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, OSDI'08 (Berkeley, CA, USA), 2008. P. 281–294.
3. *Böhm S., Běhálek M., Meca O. and Šurkovský M.* Kaira: Development environment for MPI applications. // Application and Theory of Petri Nets and Concurrency: 35th International Conference. PETRI NETS 2014. Tunis. Tunisia. June 23-27, 2014. Proceedings (G. Ciardo and E. Kindler, eds.). Cham: Springer International Publishing, 2014. P. 385–394.
4. *Pelayo F. L., Cuartero F., Valero V., Macia H. and Pelayo M. L.* Applying timed-arc Petri nets to improve the performance of the MPEG-2 encoding algorithm. // Proceedings of the 10th International Multimedia Modelling Conference, MMM '04 (Washington, DC, USA), IEEE Computer Society, 2004. P. 49.
5. *Kristensen L. M.* An approach for the engineering of protocol software from coloured Petri net models: A case study of the IETF WebSocket protocol. // Proceedings of International Workshop on Petri Nets and Software Engineering. Vol. 1160, 2014. P. 13–14.
6. *Dedova A. and Petrucci L.* From code to coloured Petri nets: Modelling guidelines. // Transactions on Petri Nets and Other Models of Concurrency VIII (M. Koutny, W. M. P. van der Aalst and A. Yakovlev eds.), Berlin. Heidelberg: Springer Berlin Heidelberg, 2013. P. 71–88.
7. *Westergaard M.* Verifying parallel algorithms and programs using coloured Petri nets // Transactions on Petri Nets and Other Models of Concurrency VI (K. Jensen, W. M. van der Aalst, M. Ajmone Marsan, G. Franceschinis, J. Kleijn, and L. M. Kristensen eds.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. P. 146–168.
8. *Voron J. B. and Kordon F.* Transforming sources to Petri nets: A way to analyze execution of parallel programs // Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, Simutools '08, (ICST, Brussels, Belgium, Belgium), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008. P. 13:1–13:10.
9. *Tarasov G. V., Kharitonov Dmitriy I.* Modeling function calls in program control flow in terms of Petri Nets // ACSIJ Advances in Computer Science: an International Journal. Vol. 3, November 2014. P. 82–91.
10. *Харитонов Д. И., Голенков Е. А., Тарасов Г. В., Леонтьев Д. В.* Метод генерации примеров моделей программ в терминах сетей Петри // Моделирование и анализ информационных систем. Том 22. № 4, 2015. С. 563-577.