СБОРНИК **НАУЧНЫХ ТРУДОВ НГТУ. – 2012. – № 2(68) –**99–104

УДК 532.783

РЕШЕНИЕ ПРОБЛЕМЫ РОСТА ДЕРЕВА СОСТОЯНИЙ МЕТОДОМ ИНКАПСУЛЯЦИИ ФРАГМЕНТОВ СЕТИ ПЕТРИ*

Д.В. ПРЫТКОВ

Предложен вариант решения проблемы роста дерева состояний, методом инкапсуляции фрагментов сети. В рамках решения предложена инфраструктура вызова/возврата, для реализации масштабируемого подход к организации раскрашенной иерархической сети Петри. Приведен пример раскрашенной иерархической сети Петри демонстрирующий особенности предложенного решения. Выполнена симуляция сети примера в среде CPN-Tools.

Ключевые слова: сети Петри, дерево состояний, инкапсуляция

ВВЕДЕНИЕ

Существующие разнообразные методологии, предлагающие свои варианты жизненных циклов (моделей) разработки программных систем, потенциально должны удовлетворять потребностям всех участников вовлеченных в процесс разработки. Специалисты в области проектирования и разработки программного обеспечения сходятся во мнении, что в современных условиях ключевую роль играет возможность реагировать на быстро меняющиеся требования. В настоящий момент наиболее распространенным подходом к проектированию и разработке ПО остается объектно-ориентированная парадигма. Современные методологии разработки практикуют итеративный подход, при этом ключевыми этапами каждой итерации являются: уточнение требований, объектно-ориентированный анализ и объектно-ориентированное проектирование. В результате, каждая итерация уточняет модель разрабатываемого программного продукта. Современным стандартом для описания моделей является язык UML, поэтому полученная модель, как правило, представлена набором UML-диаграмм. Диаграммы представляют собой удобный материал для трансляции в сеть Петри. Модель, транслированная в сеть Петри, дает инженерам инструменты для ее верификации и отладки [1,2]. Наращивая число состояний исследуемой модели можно получать результаты верификации

* Статья получена 12 января 2012 г.

Работа выполнена по заданию Министерства образования и науки РФ, проект ГК № П761 от 20.05.2010.

на более детальном уровне, однако это приводит к драматическому росту дерева состояний [2]. Увеличение числа состояний может привести к существенному снижению скорости обработки полученного дерева и даже к проблемам с построением самого дерева.

1. ПОСТАНОВКА ЗАДАЧИ

Возможное решение проблемы больших деревьев состояний может быть найдено среди столпов объектно-ориентированного программирования, а именно в инкапсуляции. Согласно определению, инкапсуляция – это сокрытие деталей реализации объектов. Очень важно ассоциировать с инкапсуляцией сокрытие от внешнего окружения внутренних данных объекта. При необходимости доступ к ним может быть предоставлен через специальные открытые функции члены данного класса. Более того, другие функции класса также должны обращаться к данным через такие специальные функции. Таким образом, учитывая особенности строения класса, с учетом требований инкапсуляции, при трансляции мы должны получить сеть, разбитую на подсети, каждая из которых будет представлена страницей в иерархической сети Петри. Каждая из подсетей фиксирует возможное поведение объектов данного класса. Конкретный объект характеризуется набором значений внутренних переменных, которые представлены в сети соответствующей переменной-маркером. Поскольку «география» вызовов всех методов и их сигнатура известны еще в момент написания кода, структура сети остается статичной, а всю динамику исполнения кода берут на себя маркеры. Таким образом, в любой момент времени маркер находится в позиции имеющей отношения к определенной функции класса. Попасть в другую функцию можно, если произойдет ее вызов, либо возврат в функцию, вызвавшую данную.

В функциональном программировании изоляция каждой функции (зависимость результата функции только от входных параметров) позволяет гарантировать правильность работы всей программы при условии правильной работы каждой ее функций в отдельности. При этом для целей тестирования нет необходимости повторять сложное окружение, выполнять предусловия, так как функции не зависят от внешних состояний. Это сводит к минимуму усилия по тестированию конкретных модулей, и позволяет гарантировать качество программы. Несмотря на то что подобный подход проблематично в полной мере перенести на императивные языки программирования (Си++, Си#, Ява), далее будет предпринята попытка локализовать его для целей проектирования приложений в стиле ООП. Возможность изоляции состояний объектов классов от внешних состояний позволит проектировать более простые с точки зрения взаимосвязей приложения, которые будет легко тестировать. С точки

зрения сетей Петри это, прежде всего, будет означать возможность деления дерева состояний на набор поддеревьев, с возможностью моделирования каждого отдельно. Это позволит справиться с проблемой чрезмерно больших деревьев состояний.

2. ПРЕДЛАГАЕМОЕ РЕШЕНИЕ

Базис решения закладывается еще на этапе проектирования. Основная идея заключается в разработке изолированных фрагментов алгоритмов, которые получают все необходимые данные в качестве входных параметров. Такой подход призван обеспечить независимость результата от внешних воздействий. В контексте сетей Петри это означает, что у изолируемой подсети существует только две позиции имеющие дуги за ее пределы.

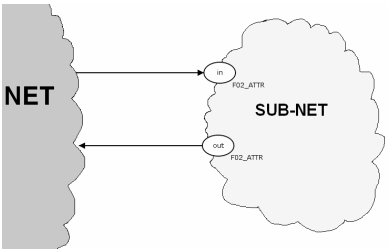

Рис. 1. Связь сети и подсети

Получив изолированную подсеть с одной входной и одной выходной (интерфейсными) позициями, мы можем проводить ее анализ независимо от остальной сети. Однако больший интерес для нас представляет обратное утверждение. Теперь можно проводить анализ сети, исключив из нее все позиции изолированной подсети, кроме интерфейсных. В результате, все узлы дерева состояний, отличающиеся только метками в позициях принадлежащих изолированной подсети, объединяются в один узел.

3. РЕАЛИЗАЦИЯ В ВИДЕ РАСКРАШЕННОЙ ИЕРАРХИЧЕСКОЙ СЕТИ ПЕТРИ

Каноническим вариантом оформление алгоритма является его реализация в виде функции. Мы, в свою очередь, будем представлять ее в виде страницы иерархической сети Петри. При этом фактические и формальные параметры, а также локальные переменные и информация о точке возврата и точке вызова будет храниться в метках определенных типов (рис. 2). Необходимо отметить,

что в предлагаемой реализации, под точкой возврата следует понимать не саму позицию, в которую перейдет метка после выхода из функции, а некоторый идентификатор, который связывает метку ***вызывающей*** функции с соответствующей меткой ***вызываемой*** функции. Проблема такого связывания возникает, например, при реализации рекурсивной функции.

```
▼EXAMPLE DECLARATION
  ▼colset ADDR=record a01:INT*a02:INT;
  ▼F01
    ▼colset F01_LOCAL=record l01:INT*l02:INT*l03:INT;
    ▼colset F01_ATTR=record addr:ADDR*loc:F01_LOCAL;
  ▼F02
    ▼colset F02_INPUT=record i01:INT*i02:INT;
    ▼colset F02_OUTPUT=record o01:INT;
    ▼colset F02_LOCAL=record l01:INT;
    ▼colset F02_ATTR=record addr:ADDR*inp:F02_INPUT*out:F02_OUTPUT*loc:F02_LOCAL;
```

Рис. 2. Объявление раскраски для хранения значений атрибутов функции

Тип ADDR используется для представления информации о точках возврата и вызова. Тип является универсальным, поэтому объявление не привязано к какой-либо функции и отсутствует специфицирующий префикс. FXX_ATTR используется для представления информации об атрибутах функции XX. FXX_INPUT, FXX_OUTPUT, FXX_LOCAL используются для представления информации о входных и выходных параметрах, а также локальных переменных функции XX, соответственно.

Для организации иерархической сети был выбран метод «*Fusion places*» (связанные позиции). В подсети ***вызываемой*** функции должны быть созданы три позиции: *init*, *in* и *out*. В нашем примере эти позиции имеют тип F02_ATTR (рис. 3). ***Вызывающая*** функция, должна содержать на своей странице позиции: *call*, *wait* и *back*, переходы *call* и *return*, а также позиции: *init*, *in*

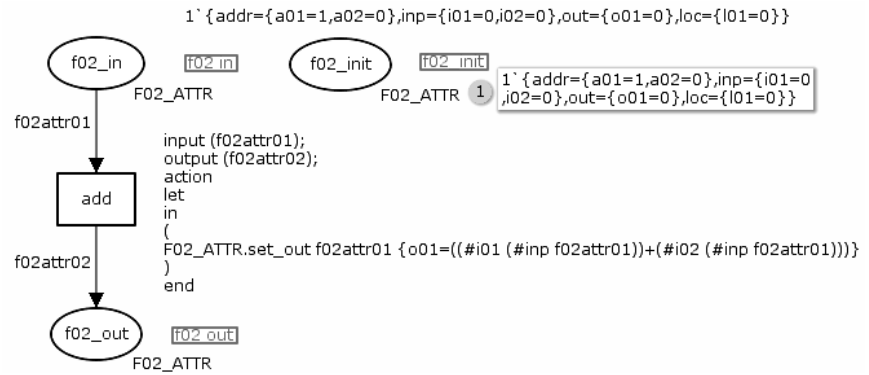

Рис. 3. Реализация функции в виде подсети

и *out* (рис. 4 и 5). В нашем примере позиции *call*, *wait* и *back* имеют тип F01_ATTR. Позиции *init*, *in* и *out* связаны с одноименными позициями в подсети ***вызываемой*** функции при помощи метода «*Fusion places*».

Для простоты примера была реализована функция F02, принимающая в качестве входных параметров, два целых числа и возвращающая их сумму. Логика сложения и размещения результата в выходном параметре сосредоточена в переходе *add* (рис. 3).

Каждому вызову функции F02 предшествует появление метки типа F01_ATTR в позиции «*call*» (вызов), принадлежащей подсети ***вызывающей*** функции. Далее, срабатывание перехода приводит к появлению метки типа F02_ATTR в позиции «*in*» (вход) подсети ***вызываемой*** функции. При этом метка типа F01_ATTR, отвечающая за хранение атрибутов ***вызывающей*** функции, переходит в позицию «*wait*» (ожидание возврата) (рис. 4).

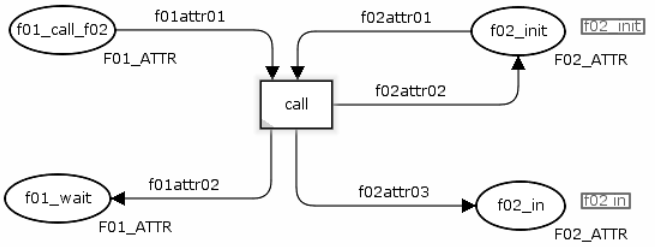

Рис. 4. Фрагмент сети ответственный за вызов функции

Позиция «*wait*» принадлежит ***вызывающей*** подсети. Позиция «*init*» (инициализация) постоянно содержит метку типа F02_ATTR с начальными значениями атрибутов ***вызываемой*** функции (рис. 3). Как видно из рис. 4, вызов функции F02 (переход *call*) приводит к появлению метки типа F01_ATTR в позиции *wait*, где она будет находиться до момента появления метки типа F02_ATTR в позиции *out* (рис. 5).

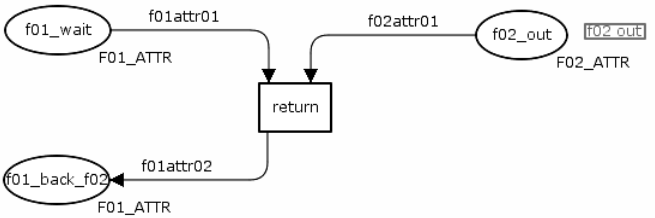

Рис. 5. Фрагмент сети ответственный за возврат из функции

Появление метки типа F02_ATTR в позиции out приведет к срабатыванию перехода return, в результате чего происходит возврат в ***вызывающую*** функцию.

ВЫВОДЫ

Важным элементом решения является принцип группировки позиций для их дальнейшей изоляции. В рамках предложенного решения объединяются позиции логически связанные границами реализуемого алгоритма. Предложенная инфраструктура вызова/возврата реализует хорошо масштабируемый подход к организации раскрашенной иерархической сети Петри для построения моделей программных продуктов. Изоляция подсети позволяет сократить дерево состояний путем объединения в один узел всех узлов отличающихся только метками в позициях принадлежащих данной подсети.

[1] *Воевода А., Прытков Д.* Применение сетей Петри на этапе объектно-ориентированного проектирования // Сборник научных трудов. – Новосибирск: НГТУ. – 2010. – № 2(60). – С. 65–76.

[2] *Коротиков С.В.* Применение сетей Петри в разработке программного обеспечения центров дистанционного контроля и управления: дис. … канд. техн. наук. – Новосибирск: НГТУ, 2007.

Прытков Дмитрий Владимирович – соискатель кафедры автоматики Новосибирского государственного технического университета. Основное направление научных исследований – автоматизация написания программного обеспечения для технических систем. Имеет 19 публикаций. E-mail: divprit@ngs.ru.

D.V. Prytkov

Conditions tree growth problem solving, by Petri net's fragments encapsulation method

The variant of a solution of a conditions tree growth problem, by net fragments encapsulation method is offered. Within the limits of the solution, offered the call/return infrastructure, for scaled approach to the organization of the colored hierarchical Petri net. The example of the colored hierarchical Petri net showing features of the presented solution is offered. Simulation of the example net in the environment of CPN-Tools is executed.

Key words: Petri nets, conditions tree, encapsulation.