РАСШИРЕННАЯ МОДЕЛЬ ДЛЯ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Старолетов С.М., Крючкова Е.Н.
Алтайский государственный технический университет им. И.И. Ползунова
(г. Барнаул)

Современные системы часто работают уже не на одном компьютере, а на их множестве, на встроенных устройствах, при этом обычно состоят из отдельных логических компонентов (под компонентом мы понимаем логическую часть системы, указываемую при проектировании на UML диаграмме развертывания), и взаимодействие между компонентами системы осуществляется через локальную сеть организации или глобальную сеть Интернет.

Работа посвящена математизации тестирования (проверки правильности) таких программных систем.

Целью исследования является создание своей адекватной математической модели распределённой недетерминированной программной системы и применение её при проведении тестирования сложных систем при помощи разрабатываемых средств.

Актуальность работы вытекает из необходимости применения различных методик при тестировании программ, перспективности применения моделей в тестировании программ, необходимости повышения культуры тестирования, отсутствия на рынке популярных средств тестирования при помощи построения моделей.

Поскольку данная работа посвящена моделированию и анализу таких систем, дадим основные определения.

Определения:

- Распределенной системой назовем систему, компоненты которой расположены на нескольких узлах сети или на одном узле с эмуляцией работы по сети. Распределенные системы могут быть реализованы как на основе стандартных средств (сокеты, вебсервисы, MPI, …), так и при помощи их комбинаций и собственных протоколов.

- Многопоточной системой назовем систему, в которой работает несколько потоков или процессов, выполняющих параллельные действия одновременно. Многопоточность в приложении может создаваться за счет явного создания потоков/процессов операционной системы, неявного создания потоков при помощи библиотеки OpenMP, работой с потоками на разных узлах с помощью средств MPI или же за счет создания легковесных потоков, встроенных в новое поколение функциональных языков.

- Недетерминированной назовем программу, действия которой в каждый момент времени носят случайный характер. Конечно, программы проектируются исходя из некоторого предположения о будущей их работе, однако сложность взаимодействующей системы может быть такой огромной, что иногда для упрощения моделирования логики работы можно считать ее недетерминированной.

Недетерминированность может возникать из-за параллельных действий клиентов в клиент-серверном приложении, возникновению событий в ответ на внешние действия, специальной недетерминированной логикой работы (например, с использованием датчика случайных чисел). При моделировании таких систем с целью упрощения возможно, принимая во внимание цели, учитывать не все возможные параметры моделируемой системы, а лишь те, которые интересны с точки зрения поставленных задач.

Современные системы, собственно, являются распределенными, многопоточными и недетерминированными. Обычно, где есть распределенность, там есть и многопоточность и недетерминированность, и такие системы — объект нашего исследования.

Всегда возможно ли математически доказать безошибочность работы таких систем? Начнем исследование данного вопроса с простых систем.

Определение: Проблема тестирования заключается в том, возможно ли для любой программы определить, выполняет ли она поставленный алгоритм или имеет ошибки и не выполняет поставленный алгоритм?

Формально, допустим что программа работает в простой однозадачной среде и определяется как набор состояний машины Тьюринга, и после каждого шага по переходу в следующее состояние осуществляется вывод значений переменных. Если после завершения программы вывод окажется в точности равен ожидаемому, то программа признается верной. Проблема тестирования состоит в том, возможно ли такую проверку

сделать для любой программы по ее описанию.

Теорема 1: Проблема тестирования неразрешима в рамках существующей теории алгоритмов.

Доказательство теоремы мы здесь не приводим, но отметим, что оно получено методом сведения неразрешимой проблемы переводимости машины Тьюринга [2] к данной проблеме.

Следует заметить, что однопоточные системы в однозадачной среде являются подмножеством более широкого множества взаимодействующих систем, и поэтому в соответствии с методом сведения [2], теорема 1 является справедливой для более широкого класса распределенных систем, а именно это можно сформировать как

Следствие из теоремы 1: проблема тестирования распределенных недетерминированных систем неразрешима в общем виде в рамках существующей теории алгоритмов.

Возникает вопрос — как же тогда осуществляется проверка правильности программ сегодня? В настоящее время обычно доказывается правильность специальных видов программ(алгоритмов) с точки зрения какого-то из методов, при этом программа может оказаться правильной с точки зрения данного метода, но неправильной с точки зрения другого. При этом скорее доказывается не правильность программы, а неправильность ее, т.е. ищется какая-либо ошибка в программе. Если таких ошибок не найдено, программа считается правильной до выявления ошибки пользователями в процессе ее эксплуатации.

Известный математик Дейкстра сказал [1]: «Тестирование программ может использоваться для демонстрации наличия ошибок, но оно никогда не покажет их отсутствие».

Данная работа посвящена выработке стратегий тестирования программ, которые помогут разработчикам находить ошибки в взаимодействующих компьютерных системах, используя специально построенную модель для таких систем. Наличие кроме самой программы её модели позволит использовать для проверки правильности дополнительный инструмент, который позволит сравнительно легко находить ошибки и узкие места в программах, анализирование которых без использования моделей было бы очень сложной задачей.

Итак, ядром исследования является математическая модель взаимодействующих систем.

Ранее в работе [4] мы рассматривали модели систем без учета их взаимодействия, после более глубокого анализа модель была существенно расширена.

Система представляет собой набор компонентов и связей между ними, а также глобальные множества отсылаемых сообщений и общих блокируемых ресурсов:

$$M = (A^*, CP(A^*), Msg, Res)$$

Здесь A^* - множество расширенных вероятностных многопоточных конечных автоматов, каждый из которых моделирует поведение компонента системы в виде взаимодействующей программы:

$$A = (s_0, S, F, \delta, E, msg \subseteq Msg, res \subseteq \mathrm{Re}\, s),$$

где S - множество состояний, $s_0 \in S$ - начальное, $F \subseteq S$ - множество заключительных состояний; E - множество событий и исключительных ситуаций; msg – подмножество глобального множества отсылаемых сообщений; res – подмножество глобального множества общих ресурсов; логика переходов описывается недетерминированной функцией переходов δ по вершинам и γ- функцией переходов по ребрам.

Определение: состояние тестируемой системы — это участок исходного кода программы (либо компонента программы), который разработчик или архитектор системы определяют как единый логический результат деятельности фрагмента программы.

Мы будем определять логическое состояние как набор строк исходного кода программной системы.

В определении понятия состояния мы предполагаем, что проект компонента системы состоит из набора файлов с исходным кодом, при этом файл состоит из набора строк с кодом:

$$Project = \bigcup_{\substack{f \subset SrcFile \\ i=1\ldots|f|}} Src_{fi}$$

Тогда состояние, с учетом того, что значимость кода состояния для работы системы определяется разработчиками, задается как:

$$s = \{ \bigcup_{i=p\ldots r} Src_{fi} \mid f \in SrcFile,\ в\ файле\ f\ с\ линии\ p\ до\ линии\ r\ код - логически\ значимый \}$$

Согласно введенному определению состояния, рассмотрим теперь, что же является переходом конечного автомата в применении к тестируемой программе.

Определение: Переход из одного состояния в программе определяется двумя

позициями в ее исходном коде: во-первых, это место в программе, в котором осуществляется какое-либо взаимодействие, с точки зрения разработчика, приводящее к изменению состояния системы (начало перехода), а во-вторых, это место начала состояния, в которое осуществляется переход(конец перехода).

Логика переходов, как уже было сказано ранее, определяется двумя функциями переходов:

$$\delta : S \times D \times P \times N \times T \times Msg \times Res \longrightarrow$$
$$2\text{^}(((T \times S)* \cup S) \times Msg^* \times Res)$$

$$\gamma : S \times S \to E^*$$

Находясь в состоянии q кратности n по действию d с вероятностью p в потоке t и по полученному сообщению msg, после снятия блокировки ресурса res модель системы недетерминировано переходит либо в несколько состояний, создав несколько новых потоков из t, или просто в следующее состояние в текущем потоке; при этом возможно возникновение и обработка конечного числа событий или исключительных ситуаций из E, отсылка сообщения и блокировка некоего ресурса.

Определение: Кратностью состояния назовем целое число N, которое показывает, что в данном состоянии может находиться одновременно не более N потоков. По умолчанию кратность состояния принимается равной 1.

Определение: Потоком в модели назовем параллельно выполняющиеся действия, заданные своей логикой работы в виде подавтоматов. Поток определяется:

- Потоком-родителем, создавший данный поток. Родитель есть у всех потоков, создаваемых в системе, кроме главного потока приложения.

- Подавтоматом, который отвечает за логику работы потока. Подавтомат определен на подмножестве состояний автомата и эти состояния и взаимодействие между ними определяют работу потока;

- Именем потока (строкой).

Таким образом,

$$T_{Name} = (T_{parent}, A' = A|_{S=S'}, Name)$$

Для анализа правильности сложных систем целесообразно было в модель программы в виде автомата добавить в функцию переходов элемент вероятности — вероятность перехода из заданного состояния в другое заданное. После этого наш автомат будет считаться вероятностным конечным автоматом.

Вероятность перехода определяется для каждой из пар состояний (состояние из которого осуществляется переход; состояние, в которое осуществляется переход на следующем шаге). Здесь можно говорить о матрице переходных вероятностей.

Возникает вопрос, откуда брать вероятности перехода из состояния в состояние и могут ли они меняться с течением времени? Ответ на этот вопрос могут дать предварительно как разработчики системы, так и данные о переходах между состояниями после реального запуска и отработки системы.

Определение: априорными вероятностями тестируемой системы назовем вероятности переходов между состояниями вероятностного конечного автомата, которые заданы разработчиками/архитекторами системы исходя из предположений о работе системы. Априорные вероятности определяют ожидаемое поведение системы.

Определение: апостериорными вероятностями тестируемой системы назовем вероятности переходов модели, каким-то образом вычисленные после работы программы на основании статистики переходов. Апостериорные вероятности определяют реальное поведение системы после её запуска.

Можно отметить, что апостериорные (вычисленные) вероятности могут быть использованы как априорные при следующем запуске моделирования.

Для моделирования работы событийной системы множество состояний разделим на

- те состояния, которые достижимы в программе в результате её последовательной работы,

- и на те, которые достижимы как модели обработчиков событий или исключений.

Определим множество E - множество, описывающие одно событие или исключение (E от англ. Event/Exception – Событие/Исключение). Событие или исключение определяется:

1. состоянием, куда будет осуществлен переход при возникновении события;

2. вероятностью того, что сработает именно данный обработчик среди всех обработчиков;

3. а также флагом типа boolean, показывающим то, что переход к обработке данного события определяет переход в так называемое ошибочное состояние:

$$E = S_E \times P_E \times W$$

Определение: Ошибочным состоянием системы назовем такое состояние, попадание в которое говорит о возникновении какой-либо контролируемой ошибки в программе с точки зрения разработчика.

Частое нахождение системы в таких состояниях говорит о ненадежности окружающей среды программы. С другой стороны, поскольку ошибочное состояние может объявляется у обработчика событий или исключений (скорее, у исключений), то описание такого состояния уже само по себе является указанием со стороны разработчика на возможность ошибочной ситуации.

Функция переходов по ребрам γ предполагает зависимость возможности возникновения события от ребра, т.е. перехода из состояние в состояние.

Кроме того, в модели имеются сообщения как элементы множества msg – средства взаимодействия между потоками и общие блокируемые ресурсы как элементы множества res. Подробнее данные элементы рассмотрены в [3].

Само взаимодействие между компонентами описывается по принципу “рукопожатия” (при рукопожатии каждый объект одновременно находится как в своем состоянии, так и в общем). В нашей модели элемент $CP(A^*)$ определяет множество точек сопряжения (попарно) у компонентов системы, а также мощности их связей (1:1,1:N), по данном принципу. Связь мощности 1:1 означает, что одному состоянию одного компонента системы соответствует ровно одно состояние другого компонента, а 1:N – что одному состоянию одного компонента соответствует N компонентов другого.

Следующим шагом исследования является формальный язык описания модели, который не зависит от используемых при создании системы языков программирования (компоненты распределенной системы могут быть написаны на различных языках) и определяет формальное текстовое описание разработанной модели.

В данном исследовании предполагается строить соотношение кода системы и модели по немного измененному автором принципу - «код и модель — одно целое». Это парадигма предполагает описание модели на языке описания моделей совместно с исходным кодом на языке программирования в одном и том-же файле. Согласно данным ранее определениям состояния и перехода, они привязаны к файлам и строкам исходного кода. Поэтому предлагается описывать модель на месте, где собственно определяются состояния, переходы, сообщения и т.д. в исходном коде.

Для того, что описание модели не вызывало бы ошибок компиляции(или интерпретации) исходных файлов, предлагается описывать модель в комментариях к исходному коду, которые будут пропускаться при компиляции кода приложения, но будут обрабатываться в целях проведения тестирования разрабатываемой анализирующей и тестирующий системой. Таким образом, код модели является метаданными к исходному коду.

Методы исследования программы и ее модели включают анализ взаимодействующих систем, формализация понятий, применение теории конечных автоматов, теории графов, теории вероятности и Марковских случайных процессов, теории формальных языков, теории алгоритмов.

Для популярных сред разработки Eclipse и Visual Studio разрабатываются расширения (Plugin, VS Package) для визуального описания моделей прямо при создании.

Собственно тестирование заключается, во-первых, в анализе модели без запуска системы (off-line), когда сложная структура модели специальным методом преобразуется в ориентированный граф и далее исследуется, и, во-вторых, с запуском системы и сравнение реальной работы системы и модели с использованием сервера тестирования (код обращения к серверу встраивается препроцессором в места описания модели).

Критерии корректности модели и системы:

А) Формальные:

- Описаны все состояния, что используются в переходах;
- не существует переходов в несуществующие состояния;
- нет состояний, в которые ничего не идет;
- матрица переходных вероятностей стохастическая;
- состояния не перекрываются;
- у сообщений есть отправитель и получатель;
- компоненты системы связаны через точки сопряжения.

Б) Тестируемые:

- Переход из состояние в состояние и обработка событий происходит строго по модели;
- кратность состояний не превышается;

- связность через точки сопряжения не нарушается;
- все сообщения доходят; блокировки срабатывают.

Таким образом, в данной статье рассмотрены проблемы тестирования современных программ, предложена модель взаимодействующей системы и применение ее в целях тестирования распределенного программного обеспечения.

Разработка поддержана грантом “УМНИК”. Ведется реализация по утвержденному плану и смете. Получено авторское свидетельство на компонент реализуемого программного комплекса “Визуальный редактор состояний”.

СПИСОК ЛИТЕРАТУРЫ

1. Гордиенко, А.В. Тестирование при оценке динамической корректности программ АСУ /А.В. Гордиенко. Программирование. 1982. №6, С. 48-52

2. Крючкова, Е.Н. Математическая логика и теория алгоритмов:Учебное пособие / Крючкова Е.Н. Алт. госуд. технич. ун-т. им. И.И. Ползунова. Барнаул, 2003. - 275 с.

3. Старолетов, С.М. Математическая модель для тестирования программ[Электронный ресурс]/С.М. Старолетов, Е.Н. Крючкова. VI Всероссийская научно-техническая конференция студентов, аспирантов и молодых ученых "Наука и молодежь − 2009". Секция «Информационные и образовательные технологии». Подсекция «Программное обеспечение вычислительной техники и автоматизированных систем». Алт. гос. техн. ун-т им. И.И.Ползунова. – Барнаул: изд-во АлтГТУ, 2009. – с 82-92. Режим доступа-
http://edu.secna.ru/publish/gorizonty_obrazovania/2009/n11/nim2009/povt2009.pdf

4. Старолетов, С.М. Тестирование распределенных приложений на основе построения моделей/Старолетов С.М., Крючкова Е.Н.//Прикладная информатика – М: Market DS publishing. – 2008. – N6. – С.124-134.