УДК: 004.41

ПРОВЕДЕНИЕ ON-LINE ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ ПОСТРОЕННОЙ МОДЕЛИ

С.М. Старолетов, Е. Н. Крючкова

Рассматривается модель современных распределенных программных систем, подход тестирования программ на основе моделей, место модели в процессе разработки ПО, а также методы динамической проверки программы по ее модели.

Ключевые слова: тестирование программ, моделирование, распределенные системы

Современные программные системы часто являются распределенными, т.е. работают уже не на одном компьютере, а на их множестве, а также на встраиваемых устройствах. При этом сложные системы 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 таких программных систем.

Тестирование программного обеспечения (ПО) — это инструмент повышения его качества с целью выявления возможных ошибок в нем [1]. Сегодня, в связи со сложностью компьютерных систем индустрия тестирования активно развивается, и наряду с программистами-разработчиками в создании системы принимают участие специалисты по тестированию (тестировщики). Однако, при усложнении системы и приеме на работу все большего и большего количества специалистов по тестированию происходит насыщение, и они уже не могут гарантировать качества безошибочной работы. Это означает, что для тестирования сложных взаимодействующих систем необходимо внедрять новые методологии.

Новым методом тестирования является подход к тестированию программ на основе построения моделей (Model based testing) [2]. При таком подходе строится математическая модель всей системы, а далее тестирование проводится по построенной модели. Хотя данный подход требует специальных знаний у тестировщика, уже сам факт перехода к моделированию в тестировании означает серьезное отношение к качеству 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

Математическая модель распределенной системы

В данной работе предлагается многоуровневая модель системы, описывающая многокомпонентные распределенные системы, каждый компонент которой может обеспечивать параллельные действия.

В предлагаемой математической модели можно выделить следующие уровни абстракции:

Система, как множество взаимодействующих расширенных конечных автоматов – автомат специального вида

Расширенный вероятностный конечный автомат – модель компонента системы

Каждый автомат состоит из набора состояний, переходов и операций

Подавтомат — автомат, определенный на подмножестве состояний

Состояние, привязано к коду программы.

Рассмотрим модель распределенной системы:

$$System = \begin{pmatrix} Node, A^*, A_{cooper}, BoundNodes, \\ Msg, Res, CP(A^*) \end{pmatrix}$$

Здесь:

Node – множество узлов, на которых работает распределенное приложение,

$Node = \{(NodeName,\ NodeAdress\ ,NodeType\ )\}$ - кортеж, содержит информацию об узле: NodeName – строковое имя узла, NodeAdress

– адрес протокола IP или DNS имя, по которому узел доступен в сети;

$NodeType = \begin{Bmatrix} WindowsPC, LinuxPC, \\ Mobile, SmartPhone \end{Bmatrix}$ - множество различных типов систем, на которых могут работать узлы (ПК под управлением Windows, Linux, мобильный телефон, смартфон – возможны и другие типы);

$BoundNodes : Node \to AN, n \in N,$
$AN = \{(A, n)\}$ - отображение, определяет для каждого узла, какие компоненты, заданные в виде расширенного конечного автомата A ($A \in A^*$), выполняются на данном узле, и с какой кратностью (на узле может работать n одинаковых компонентов системы в виде расширенного автомата A);

Msg – глобальное для системы множество посылаемых и получаемых сообщений;

Res – глобальное множество разделяемых ресурсов, к которым имеют доступ все компоненты системы. Поскольку ресурсы являются разделяемыми, одновременный доступ к ним может привести к ошибкам, и необходимы специальные методы синхронизации.

A_{cooper} - специального вида автомат, моделирующий порядок взаимодействия в распределенной недетерминированной системе. Состояния автомата — это компоненты системы с заданной кратностью, а переходы — взаимодействия между ними как результат отсылки и ожидания сообщения.

$A_{cooper} = (Cn, Cn_{0,} Cn_f, \delta_{cooper}, In, \Sigma)$

Здесь:

Cn – множество состояний автомата A_{cooper} , при этом состояние - это взаимодействующий компонент системы с учетом кратности, т.е. $Cn = \{(A, n_+)\}$, где n_+ - суммарная по всем узлам сети кратность компонента системы, заданного расширенным автоматом A .

$Cn_0 \subseteq Cn$ - подмножество начальных состояний, с каждого из которых может начаться взаимодействие.

$Cn_f \subseteq Cn$ - подмножество конечных состояний, после отсылки сообщений в которые система заканчивает работу.

In – отображение, определяет возможность приема сообщений компонентом от нескольких источников

$In : Cn \to Cn^* \times InType,$
$InType = \{\& - receive, \wedge - receive\}$,

т.е. для состояния из Cn может быть указана возможность ожидания сообщений от нескольких состояний по типу AND (ждать сообщения от указанных компонентов) или OR (ждать сообщения от одного из указанных компонентов).

Автомат действует согласно недетерминированной функции переходов δ_{cooper} : $\delta_{cooper} : Cn \times \Sigma \times In \to 2^{Cn}$, при этом $c_1 \to c_2$ означает, что c_1 инициирует отсылку сообщения, а c_2 - ждет сообщения (или сообщений по типу AND и OR).

Следует учесть, что взаимодействующие клиенты могут иметь различные кратности (пример — один сервер и несколько клиентов). В таких случаях можно говорить о мощностях связей 1:1, 1:N, M:N. При этом можно представлять взаимодействие с точки зрения одного клиента, а если есть какое-то межклиентское взаимодействие, то делать петлю в данном состоянии.

Σ - алфавит автомата, задают словесные действия — надписи на переходах («что это за сообщение»).

Рассмотрим процесс моделирования по модели распределенной системы. На каждом шаге недетерминированно выбирается взаимодействие и осуществляется переход согласно δ_{cooper} . При попадании в заключительное состояние процесс взаимодействия завершается.

Перейдем к модели на более низких уровнях абстракции.

Каждый компонент системы моделируется в виде расширенного вероятностного многопоточного конечного автомата с возможностью отправки и приема сообщений между состояниями и блокировкой ресурсов:

$A = (s_0, S, F, \delta, \gamma, E, msg \subseteq Msg, res \subseteq Res)$,

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 описыва-

ется недетерминированной функцией переходов δ.

Любой объект, в том числе и программа, характеризуется состоянием и поведением. Состояние объекта – это ситуация в его жизни, на протяжении которой он удовлетворяет некоторому условию, осуществляет определенную деятельность или ожидает какого-то события. Автомат, находясь в некотором состоянии, получает внешнее событие и переходит в новое состоянии в соответствии с функциями переходов.

Состояние компонента системы – определенная разработчиком системы последовательность линий кода, которая, по его мнению, отражает неделимое состояние системы в каждый момент времени:

$$S = \{\underset{i=p..r}{\cup} S_{fi} \mid f \in SrcFile\}$$

Здесь объединение ведется по строкам исходного кода программы.

Логика работы расширенного автомата A в общем случае определяется функциями переходов:

- недетерминированной функцией переходов δ, она задает отображение: $\delta : S \times D \times P \times N \times T \times Msg' \times Res' \longrightarrow 2^{\wedge}(((T \times S)^{*} \cup S) \times Msg'^{*} \times Res')$

  Находясь в состоянии из S кратности N по действию из D с вероятностью из P в потоке из T и по полученному сообщению из Msg' после установки блокировки ресурса на Res' модель компонента системы либо переходит в несколько состояний, создав несколько новых потоков из T, или просто в следующее состояние в текущем потоке; при этом возможны отсылка сообщения из Msg' и разблокировка некоего ресурса из Res'.
- Функцией переходов «по ребрам» γ, определяющей возможность возникновения некоторого числа событий или исключительных ситуаций при переходе из состояния в состояние $\gamma : S \times S \to E^{*}$.
- Операцией редукции, осуществляющий сворачивание потоков в один:

$$join : (S \times T)^{+} \to S \times T$$

Эту операцию нельзя выразить в функции перехода, поскольку мы рассматриваем локальные функции перехода в зависимости от текущего состояния и потока, а редукция осуществляется в зависимости от нескольких состояний и потоков.

Все действия в расширенном автомате связаны с некоторой вероятностью, и здесь мы говорим о моделировании потока управления (control flow), но не потока данных (data flow).

Место модели в процессе разработки

Для описания модели был разработан независимый от целевого языка программирования (на котором написан компонент моделируемой системы) язык описания модели в виде вложенных XML сущностей. Например, в тег «состояние» вложено большинство тегов, таких как «переход», «создание потока», «ожидание сообщения» - они действуют в конкретном состоянии и там же описываются.

Следующим шагом необходимо “привязать” модель к исходному коду. В данном исследовании предполагается строить соотношение кода системы и модели по измененному принципу - «код и модель — одно целое». Это парадигма предполагает описание модели на языке описания моделей совместно с исходным кодом на языке программирования. Согласно нашему определению 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 на разработанном языке не вызывало бы ошибок компиляции (или интерпретации) исходных файлов, предлагается описывать модель в псевдокомментариях к исходному коду, которые будут пропускаться при компиляции кода приложения. Таким образом, код модели является метаданными к исходному коду.

Для модели высокоуровнего взаимодействия (автомат A_{cooper}) модель описывается в графическом виде на основе разрабатываемых программных средств (подключаемые модули к средам разработки). Следует разграничить действия по описанию модели взаимодействия высокого уровня архитектором системы и моделей более низкого уровня разработчиками компонентов такой системы.

Виды тестирования по модели

Тестирование предлагается проводить двумя способами: статическое и динамическое.

- Статическое (off-line) тестирование по модели означает полную замену систему

её моделью. С реальным кодом системы, так и с выполняемыми компонентами, получившимися в результате компиляции исходного кода, работа не производится. На основе вероятности переходов можно промоделировать работу системы и взаимодействия компонентов. Для статического тестирования также можно применять методы теории графов и Марковских случайных процессов для анализа характеристик модели, достижимости состояний и генерации тестовых наборов.

- Динамическое (on-line) тестирование проводится на скомпилированной рабочей системе с целью проверки её поведения относительно построенной модели. Рассмотрим данный процесс подробнее.

Организация процесса on-line тестирования по модели

В данном случае предлагается использовать выделенный тестирующий сервер, получающий через TCP/IP соединение данные о всех происходящих в модели событиях. При этом на сервере воссоздается динамическая модель работающей системы, которую можно сравнить с ожидаемой и найти ошибки.

Для обеспечения отсылки данных о проходящих событиях в модели необходимо добавлять к коду компонента системы специально сгенерированный код на целевом языке программирования в точках описания модели, который будет взаимодействовать с тестирующим сервером в процессе работы компонента, посылая все события на сервер, где они будут регистрироваться в базе данных. Вставка дополнительного кода осуществляется препроцессором при сборке проекта.

В процессе тестирования после разворачивания тестируемой системы по тестовым узлам, запуска сервера и начала взаимодействия, на сервере начинается построение динамической модели системы (рисунок 1).

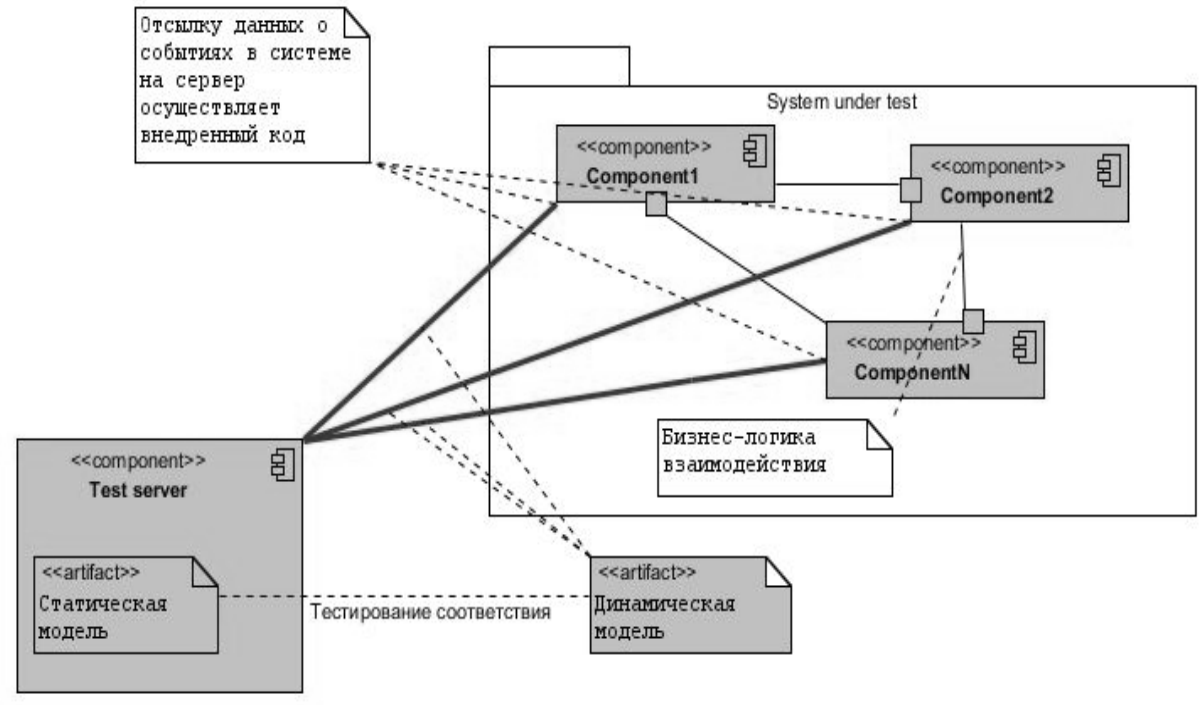

Рисунок 1 - Архитектура on-line тестирования

При этом выполняется тестирование (conformance-testing), включающее:

- Переход из заданного состояния в одно из заданных.

Переходы из состояния в состояние производятся строго по модели. Осуществление перехода в состояние, которое не ожидается в данный момент согласно функции переходов автомата, говорит об ошибке в потоке управления.

- Срабатывание событий и исключительных ситуаций.

При переходе из состояния в состояние может осуществляться генерация событий и исключительных ситуаций. События могут происходить при невозможности отправить сообщение, тайм-ауте при ожидании освобо-

ждения ресурса. При этом они должны происходить в ожидаемых разработчиками местах.

- Соответствие модели верхнего уровня моделям нижнего уровня по сообщениям и кратностям.

Модель верхнего уровня (A_{cooper}) описывает возможные взаимодействия через отправку/прием сообщений, модели компонентов более низкого уровня описывают обмен конкретных сообщений между заданными состояниями. Если заданный порядок обмена нарушен — это ошибка во взаимодействии.

- Правильность создания потоков, кратности состояний

При создании потока и моделировании этого действия на низком уровне абстракции проверяется число ожидаемых и созданных потоков. Превышение заданной кратности для состояний говорит либо о неправильности потока управления, либо о возможных высоких нагрузках.

- Правильность отсылки и приема сообщений

Для любого отправленного сообщения проверяется, дошло ли оно до получателя и правильно ли обрабатывается потеря сообщения.

- Критические секции, связанные с блокировкой внешних ресурсов

Если производится блокировка ресурса одним потоком /компонентом и производится попытка доступа к нему другим потоком/компонентом, то проверяется, что доступ к ресурсу реально имеет только его владелец.

- Возникающие дедлоки — бесконечные ожидания (deadlocks).

Дедлок — это бесконечное ожидания объектом Obj1 освобождения ресурса, когда захвативший этот ресурс Obj2 сам ожидает освобождения Obj1. Цикл в графе ожидания ресурсов сигнализирует о наличии дедлока.

- Связность компонентов через точки сопряжения (соответствие состояний, в каких находится один компонент, состояниям, в которых находится другой компонент).

Если определены точки сопряжения, то проверяется, действительно ли выполняется соответствие связанных состояний.

- Возможность отслеживать для каждого действия так называемый «state trace» – те состояния и события, которые уже произошли перед данным событием — при обнаружении несоответствия это позволит исследовать его причину.

Кроме того, на тестирующем сервере производится сбор статистики работы системы и вычисление апостериорных вероятностей всех возникающих действий, которые потом можно использовать как «априорные» вероятности при имитационном моделировании. Также на сервере собирается статистика по наиболее часто совершаемым переходам и использованию межкомпонентных связей и задержек при взаимодействиях, что позволит помочь в возможном решении проблем с «узкими местами» в системе.

Заключение

Таким образом, в статье рассмотрены проблемы тестирования современных программ, предложена модель взаимодействующей системы и применение ее в целях тестирования распределенного программного обеспечения. На наш взгляд, использование моделирования в тестировании ПО ведет к общему повышению культуры тестирования и позволит находить ошибки в особо сложных системах.

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

СПИСОК ЛИТЕРАТУРЫ

1. Котляров, В.П. Основы тестирования программного обеспечения: Учебное пособие/ В.П. Котляров, Т.В. Коликова.- М.: Интернет Университет Информационных Технологий; Бином, 2006. — 285с.
2. Ibrahim, K. El-Far and James A. Whittaker. Model-based Software Testing /Encyclopedia on Software Engineering (edited by J.J. Marciniak) - Wiley, 2001 [Электронный ресурс] URL: - http://www.geocities.com/model_based_testing/ModelBasedSoftwareTesting.pdf

*К.ф.-м.н., **Крючкова Е.Н.** - (3852) 36-75-83, Алтайский гостехуниверситет; аспирант **Старолетов С.М.**, тел. (3852) 62-98-23, sergey.staroletov@gmail.com - Алтайский гостехуниверситет.*