Совместное применение методов Model driven developing и Model based checking

Старолетов Сергей Михайлович
Алтайский государственный технический университет им И.И. Ползунова
г. Барнаул, РФ
serg_soft@mail.ru

***Реферат* — в статье описывается основная архитектура своего решения по объединению подходов разработки и тестирования на основе моделей.**

Ключевые слова — MDD; MBT; тестирование; верификация; распределенные системы

I. Введение

В последнее время в Computer Science все чаще и чаще применяется Model Checking[1] для верификации сложных взаимодействующих систем. С другой стороны, при разработке современных приложений после появления UML и CASE средств стали популярны методы Model Driven Development, когда разработка проводится с предварительным моделированием. Соответственно, разумным будет объединить данные подходы в единый цикл разработки, который будет получать преимущества проектирования программного обеспечения с тестированием математическими методами.

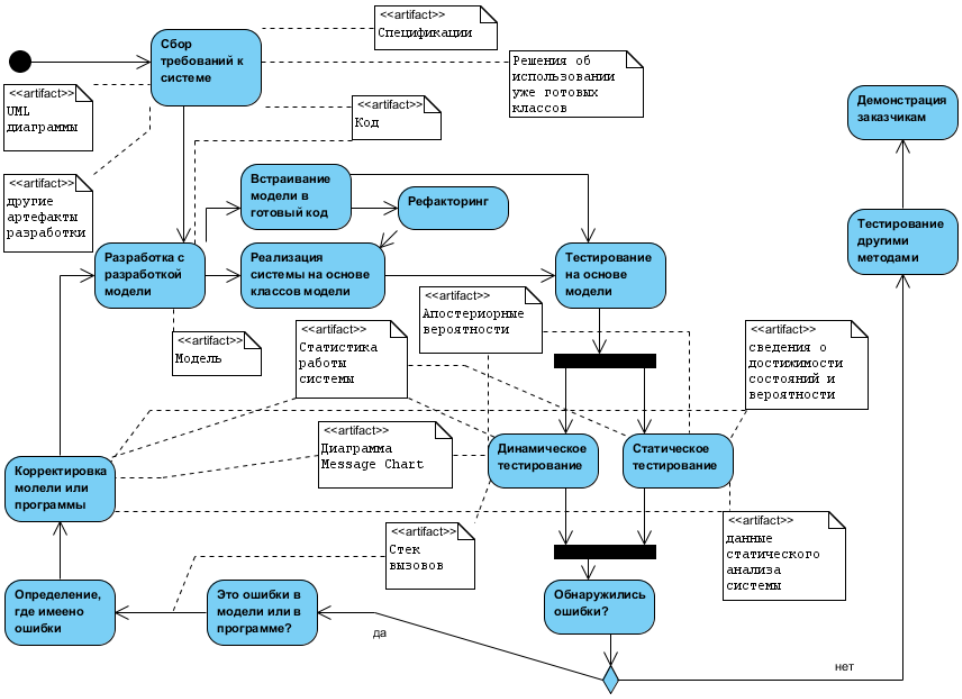

Рис. 1. Предлагаемый процесс разработки

В исследовании предлагается разрабатывать программы на разных уровнях абстракции в виде взаимодействующих расширенных автоматов, причем как с нуля, так и строя модель по коду, который не обязательно должен быть построен с помощью автоматных конструкций.

Отличием предполагаемого подхода от других следует считать то, что в настоящей работе была сделана попытка предложить полный процесс разработки и проверки правильности системы с использованием модели на разных уровнях абстракции и с использованием разной степени готовности исходного кода системы.

II. Постановка задачи

- Разработать автоматную стохастическую мультиагентную (многокомпонентную) модель, позволяющую представлять современную программную систему на разных уровнях абстракции.
- Разработать методологию внедрения математической модели в процесс проектирования программных систем.
- Разработать алгоритмы и прототипы программных средств для проведения динамического тестирования соответствия исходной модели и модели, построенной динамически в процессе работы программной системы.
- Разработать алгоритмы и прототипы программных средств для проведения статической верификации программной системы по модели.

III. Процесс разработки

Рассмотрим предполагаемый процесс разработки (рисунок 1). Первоначально разработка или итерация разработки начинается со сбора требований к исходной системе. Здесь могут быть использованы спецификации и UML диаграммы, исходя из которых можно говорить о примерной структуре создаваемой модели для системы. После выяснения требований происходит разработка системы (написание кода, модульное тестирование, и одновременное описание модели). После завершения разработки необходимо провести тестирование и верификацию реализованной функциональности.
Тестирование на основе модели предлагается проводить двумя способами:

- Статическое тестирование (верификация и симуляция) по модели, когда мы имеем модель системы и в процессе проверки соответствия работаем только с моделью.
- Динамическое тестирование проводится на скомпилированной рабочей системе, с целью проверки её поведения относительно построенной модели.

IV. Связь кода и модели

Если система или компонент системы реализуется с нуля, то можно воспользоваться предлагаемой объектно-ориентированной моделью для описания расширенных автоматов и их связей. Если же требуется построить модель по существующему коду и код не предполагается переписывать, то необходимо встраивать в код описание модели на специальном языке в виде XML тегов (рисунок 2). При этом *состояние* можно считать за участок линий кода, выполняющий некоторые общие действия, а переход в другое состояние снабдить некоторой вероятностью (чтобы промоделировать возможные действия пользователю по выбору вариантов продолжения и изменения значений ключевых переменных при переходе).

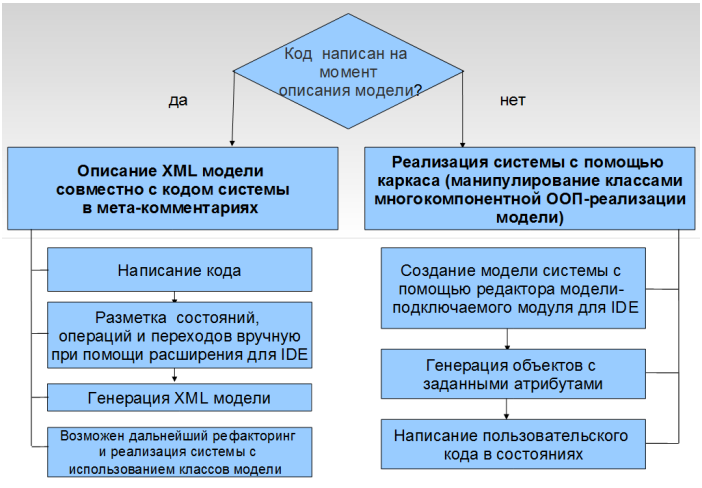

Рис 2. Код и модель

V. модель

Модель программной системы задается на разных уровнях.

На *верхнем уровне* – это структурный автомат, содержащий логику поведения (т.е. межкомпонентное взаимодействие), а также глобальные множества общих ресурсов и сообщений. Структурный автомат описывает поведение сложной системы в общем, какой компонент в каком порядке должен осуществить взаимодействие с другим компонентом. Такая модель может быть получена из диаграмм развертывания и применением подхода Р. Бара[2] на основе структурных схем (если каждый компонент системы считать процессом) для проектирования системы.

На *нижнем уровне* каждый компонент описывается в виде расширенного вероятностного конечного автомата заданного вида. При проведении исследования[3] изначально модель описывалась как автомат со сложной функцией переходов, далее она была преобразована в описание автомата с состояниями, переходами и операциями, что позволяет перейти к объектно-ориентированному определению модели и описанию модели посредством XML языка.

Логику работу программы можно промоделировать в виде конечного автомата. Однако, требуется его расширение до тех возможностей, которые мы будем проверять в процессе тестирования и верификации. В процессе исследования была предложена модель в виде расширенного вероятностного автомата с описанием потоков в виде подавтоматов и добавлением операций синхронизации. Вероятностный автомат необходим для моделирования ситуаций в целевой системе, когда переход из состояния в состояние может осуществляться в результате некоторых условий, а мы эти условия не хотим учитывать, поскольку важен факт именно осуществления взаимодействия. При проектировании системы такие вероятности могут быть предварительно оценены разработчиками, а позже уточнены в процессе сеансов динамического тестирования.

Рассмотрим представление модели нижнего уровня в виде состояний, переходов и операций. В работе[3] первоначально было построено описание автоматной модели в виде расширенного конечного автомата со сложными функциями переходов. Однако далее такая модель была преобразована в другой вид, который позволяет проще перейти к объектно-ориентированной модели (рисунок 3). Формально, модель нижнего уровня для компонента системы в виде расширенного конечного автомата определяется как тройка:

$$A = (S, Trans, Op)$$

где $Trans = S \to S \times E^*$ – отображение, определяет переход из состояния в состояние с возможной генерацией событий или исключений, $Op = \{fork, join, send, receive, block, unblock\} \times E^*$ - множество рассматриваемых ниже операций (сложных

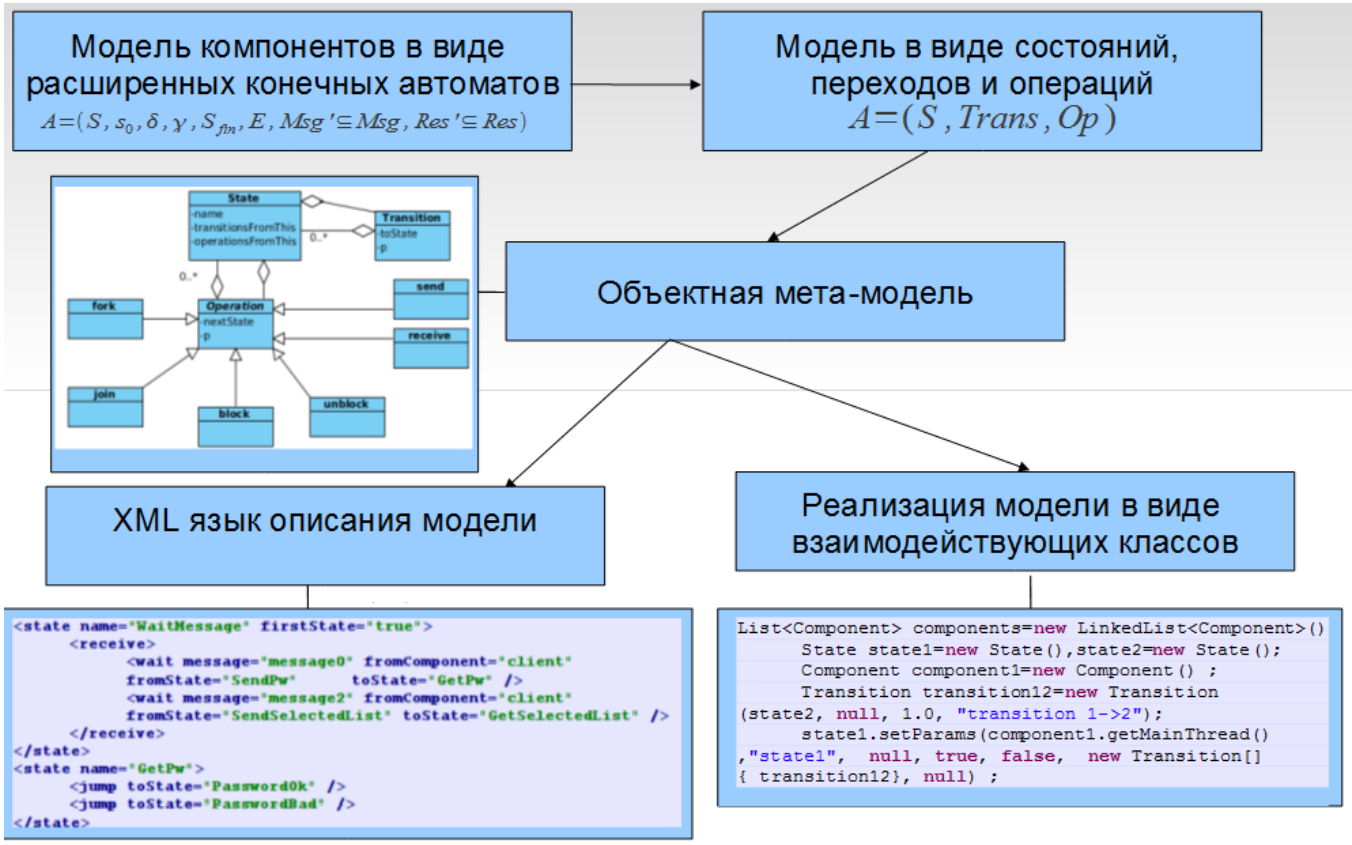

Рис. 3. Различные представления модели нижнего уровня, а также пример XML языка описания модели и сгенерированный код системы переходов созданной модели в виде классов

переходов) в расширенном автомате также с возможными связанными событиями.

Операции:

- Создание потока $fork : P \times S \to (S \times T)^+$. Находясь в некотором состоянии из S, с некоторой вероятностью P, компонент переходит в несколько состояний (хотя бы в одно), создав несколько потоков (элементы множества T, при этом текущий поток является родителем для вновь созданных потоков и продолжает выполняться вместе с ними).
- Ожидание завершения потоков $join : P \times (S \times T_{parent}) \times (S_{fin} \times T_{slave})^+ \to S \times T_{parent}$

  Здесь $T_{parent} \in T$ – родительский поток, который осуществляет ожидание подчиненных потоков $T_{slave} \subseteq T$. Более кратко это можно записать как $join : P \times (S \times T)^+ \to S \times T$. Находясь в некотором состоянии, в некотором потоке, с вероятностью P компонент начинает ожидать завершения некоторого количества потоков из множества T (перехода их подавтоматов в заключительное состояние), и после этого переходит в новое состояние из S в своем потоке.
- Посылка компонентом сообщения $send : S \times T \times P \to (S \times Msg) \vee E$.

  Компонент, находясь в состоянии из S в потоке из множества T с некоторой вероятностью P, инициирует отсылку сообщения из множества Msg, при этом переходит в новое состояние из S или, при неудачной попытке отсылки, инициирует событие из множества E.
- Получение компонентом сообщения $receive : S \times T \times P \times Msg \to S \vee E$.

  Компонент, находясь в состоянии из S в потоке из множества T с некоторой вероятностью P, начинает ожидать сообщение Msg и, получив его, переходит в новое состояние из S. При ошибке получения сообщения (например, по таймауту), генерируется событие из множества E.

- Блокировка разделяемого ресурса $block : S \times T \times N \times P \times \mathrm{Re}s \rightarrow (S \times 1 \times \mathrm{Re}s) \vee E$

  Находясь в состоянии из S в потоке из T с некоторой кратностью N с вероятностью P, поток текущего компонента начинает блокировку ресурса из $\mathrm{Re}s$ и при удачной блокировке переходит в новое состояние с кратностью 1. При неудачной блокировке поток зависает и ждет освобождения ресурса, при неуспешном ожидании может быть сгенерировано событие из E.

- Разблокировка разделяемого ресурса $unblock : S \times 1 \times \mathrm{Re}s \times P \rightarrow (S \times N \times \mathrm{Re}s) \vee E$

  Находясь в состоянии из S, владея заблокированным ресурсом $\mathrm{Re}s$ и имея кратность состояния 1 (никакой другой поток не находится в данном состоянии), компонент с некоторой вероятностью P осуществляет попытку разблокировки ресурса $\mathrm{Re}s$, при успешной разблокировке переходит в новое состояние с любой допустимой кратностью, при неуспешной – может генерировать событие из E.

Имея данную модель, можно моделировать взаимодействующие многопоточные системы, построенные на основе выделения состояний. На рисунке 3 также приведены шаги по переходу между различными представлениями модели и пример кода сгенерированного компонента, основанного на состояниях.

VI. Статическая верификация

В настоящее время сообществами разработаны мощные способы проверки моделей на основе теории Model checking, разработаны различные верификаторы. Однако существует проблема применимости языков описания моделей для таких верификаторов к реальным программным системам. Требования к верифицируемым свойствам системы обычно задаются в виде предикатов темпоральной логики и мало понятны для разработчика или тестировщика. В статье [4] показаны способы построения эквивалентной модели на языке Promela для верификатора Spin[5] по рассматриваемой автоматной стохастической модели. После такого преобразования Spin управляется тестирующей средой и получает на вход сгенерированные по целевой системе модели на необходимом языке и сгенерированные предикаты для проверки, после чего тестирующая среда обрабатывает результаты верификации, поступающие от верификатора.

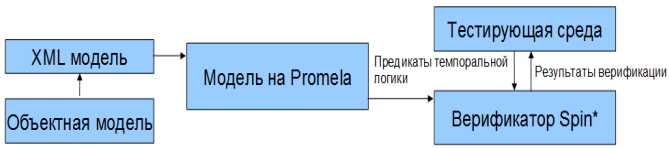

Рис 4. Схема перевода моделей при статической верификации

Статически целесообразно верифицировать

- логику системы верхнего уровня,
- правильность синхронизации посредством сообщений, корректную работу с ресурсами,
- достижимости состояний,
- соответствие нахождения в заданных состояниях одного компонента относительно состояний другого.

VII. Динамическое тестирование

Динамическое (on-line) тестирование предполагает построение динамической модели ($M_{\text{дин}}$) работающей системы и сравнение ее с описанной моделью ($M_{\text{стат}}$), причем если $M_{\text{дин}} \subseteq M_{\text{стат}}$, то данный сеанс тестирования прошел успешно, иначе модели не соответствуют друг другу, что связано либо с найденными ошибками в программе, либо с ошибками в описании модели. Динамическое построение модели позволяет отследить момент начала несоответствия. Комплексное построение модели всех компонентов системы осуществляется динамически на сервере системы тестирования с использованием протокола TCP/IP.

- Если система реализуется с использованием предлагаемых классов модели, то они уже имеют функциональность по отсылке данных для построения модели на сервере тестирования.
- Если модель дописывается в код существующей системы, то препроцессором при построении проекта добавляются конструкции на целевом языке реализации компонента для отправки данных о событиях в модели на сервер.

Динамическим тестированием проверяется:

- переходы, события, исключения согласно модели;
- передача и прием сообщений согласно модели;
- создание потоков, кратности, блокировки ресурсов.

Инструменты, получаемые в результате динамического тестирования:

- Statetrace (переходы и события в модели).
- MSC (диаграмма посылаемых сообщений).
- Апостериорные вероятности переходов.

- Статистика по посещаем состояниям, времени переходов и взаимодействий.

VIII. Состояние реализации и дальнейшая работа

В настоящий момент описанная концепция реализована в виде прототипов как расширения средств разработки Visual Studio и Eclipse. ПО позволяет:

- создавать и редактировать состояния расширенного автомата из среды разработки;

- генерировать код модели на разработанном формальном языке, а также код взаимодействующей системы на языке Java по спроектированной модели, пользователю остается написать код логики системы в выделенных состояниях;

- проводить динамическое тестирование по модели с использованием тест-сервера;

- осуществлять интерфейс модель-верификатор из среды разработки и отображать результаты верификации.

Подход используется при проектировании программных систем по дисциплине “Проектирование сетевых и многопоточных приложений”, когда студенты проектируют взаимодействующие системы, начиная со структурного автомата, далее проводят моделирование с описанием расширенного автомата, а потом реализуют сетевое и межпроцессорное взаимодействие.

В дальнейшем хотелось бы перейти к более строгой верификации за счет внедрения технологий верифицирующей компиляции, более плотной работе с исходными кодами верификаторов, унификации подхода с другими новыми подходами в Model based Checking, Testing, Developing.

Используемые источники

[1] Ю. Г. Карпов. Model Checking. Верификация параллельных и распределенных программных систем. СПб.: БХВ-Петербург. 2009. 560 стр.

[2] Р. Бар "Язык Ада в проектировании систем". М.: Мир. 1988.

[3] С. М. Старолетов, Е. Н. Крючкова. Моделирование распределенных многокомпонентных программных систем и их тестирование на основе автоматных вероятностных моделей. Барнаул: изд-во АлтГТУ. 2011. 107 стр.

[4] S. Staroletov. Model of a program as multi-threaded stochastic automaton and its equivalent transformation to Promela model. Ershov informatics conference. PSI Series, 8th edition. International workshop on Program Understanding. Proceedings. Novosibirsk. 2011. pp. 33-38

[5] Spin online references. http://spinroot.com/spin/Man/