На правах рукописи

УДК 004.054

Старолетов Сергей Михайлович

Моделирование распределенных недетерминированных программных систем и их тестирование на основе автоматных мультиагентных вероятностных моделей

Специальность 05.13.11 –
Математическое и программное обеспечение
вычислительных машин, комплексов и компьютерных сетей

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата физико-математических наук

Новосибирск 2011

Работа выполнена в ФГБОУ ВПО «Алтайский государственный технический университет им. И.И. Ползунова»

Научный руководитель: профессор, кандидат физико-математических наук
Крючкова Елена Николаевна

Официальные оппоненты: доцент, доктор технических наук
Рояк Михаил Эммануилович

доцент, кандидат физико-математических наук
Рубан Анатолий Альбертович

Ведущая организация:

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Защита состоится 26 декабря 2011г. в 15.00 на заседании диссертационного совета ДМ 003.032.01 в Институте систем информатики им. А.П. Ершова Сибирского отделения РАН по адресу:
630090, Новосибирск, пр. Академика Лаврентьева, 6.

С диссертацией можно ознакомиться в читальном зале библиотеки ИСИ СО РАН (пр. Академика Лаврентьева, 6).

Автореферат разослан _____ ноября 2011 г.

Ученый секретарь диссертационного совета ДМ 003.032.01  к. ф.-м.н. Мурзин Ф.А.

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

В настоящей диссертационной работе рассматриваются модель, методы и инструментальные средства для проектирования и тестирования программных систем для параллельной и распределенной обработки данных согласно принципам MDD (разработка, управляемая моделями) и MBT (тестирование на основе моделей).

Как известно, в процессе разработки тестирование полученного продукта занимает большую часть времени. Модель программной системы, используемая в процессе ее разработки с применением принципа MDD, может быть применена для проверки ее правильности, если для описания модели путем декомпозиции реальной системы изначально были выбраны параметры, адекватно описывающие проверяемое состояние системы.

В исследовании предлагается и исследуется математическая модель многокомпонентных распределенных программных систем, предназначенная для описания таких систем на разных уровнях абстракции: от высокого уровня взаимодействия между компонентами системы, до низкого уровня на основе выделения состояний в коде. Распределенная система рассматривается как набор расширенных вероятностных многопоточных конечных автоматов с обработкой событий и исключений, с возможностью обмена сообщениями как внутри, так и между компонентами, а также с захватом и освобождением общих ресурсов.

Разработаны методы описания модели для программной системы по принципу «код и модель-одно целое» в двух вариантах:

- при наличии реализованного программного кода описание модели на разработанном языке встраивается в код системы (описание модели по коду);
- реализация программной системы при помощи объектно-ориентированной реализации модели в виде классов (проектирование системы как модели).

Тестирование программной системы по ее модели представляет собой как динамическую проверку соответствия модели и реальной системы, так и статический анализ модели без реальной системы.

Динамический метод тестирования заключается в воссоздании на сервере тестирования модели системы в процессе ее работы, проверке переходов в состояния и других параметров согласно заданной модели.

Статический метод тестирования включают в себя имитационное моделирование, отладку, проверку достижимости состояний, вычисление вероятностей переходов в системе по ее модели.

Объект исследования — современные многокомпонентные распределенные программные системы, многопоточные системы, системы, построенные на обмене сообщениями.

Актуальность темы. С ростом сложности программных систем возни-

кает проблема обеспечения достаточного уровня надежности разрабатываемого ПО, ошибки в котором могут нанести серьезный экономический ущерб и привести к жизненно-опасным ситуациям. Современные технологии программирования не могут обеспечить эффективных методов безошибочного проектирования ПО. В настоящее время на рынке нет удобных и эффективных продуктов для тестирования программ с использованием математических моделей, как нет и общепризнанных математических моделей для описания многокомпонентных распределенных систем. Большая работа проделана в исследовательском центре корпорации Microsoft профессором Ю. Гуревичем, ведутся исследования в научных центрах NASA и Bell labs, в России следует отметить работы института системного программирования РАН и кафедры технологий программирования СпбГУ ИТМО. Однако, практическая применимость для тестирования предлагаемых моделей в компаниях по разработке ПО невелика, что является следствием сложности предлагаемых моделей и низкой степенью их вовлечения в реальные процессы разработки. В связи с этим актуальным является комплексное исследование предметной области в сфере сложных многокомпонентных взаимодействующих программ, их моделирование и практическое применение построенной модели для верификации и тестирования ПО.

Целями диссертационной работы являются: развитие теории программирования на основе конечных автоматов с параллельной и распределенной составляющей, программная реализация модели и методов ее описания, направленная на практическое внедрение использования моделей в процессе проектирования и анализа программных систем различного назначения..

Основными требованиями к модели являются:

- адекватное описание различных классов распределенных программных систем;
- возможность внедрения фазы описания модели в процесс разработки программного обеспечения;
- возможность проведения тестирования программных систем на основе разработанной модели динамическими и статическими методами.

Научная новизна:

1. Разработана автоматная стохастическая мультиагентная математическая модель, позволяющая представлять современную программную систему на разных уровнях абстракции.
2. Разработана методология внедрения математической модели, описанной на формальном языке на основе объектной декомпозиции, в процесс разработки программных систем, а также реализация способов проектирования ПО на основе представления объектно-ориентированной модели в виде классов.
3. Разработаны алгоритмы для проведения динамического тестирования соответствия заранее определенной модели и модели, построенной в процессе работы программной системы с использованием тестирующего сервера.
4. Разработаны алгоритмы для проведения статической верификации программной системы по модели.

Практическая значимость. На основе результатов исследования в виде алгоритмов и модели созданы прототипы комплекса программ, предназначенные для проведения разработки и тестирования по модели и включающие следующие компоненты:

- объектно-ориентированную реализацию модели в виде классов на языке Java;
- инструменты для описания модели по принципу «код и модель – одно целое»;
- инструменты для генерации кода на разработанном языке описания модели и синтаксических конструкций на языке программирования реализуемого компонента системы;
- препроцессор для вставки при компиляции промежуточного кода, предназначенного для динамического тестирования по модели;
- сервер динамического тестирования и средства анализа результатов динамического тестирования;
- средства для статического тестирования по модели.

Компоненты интегрируются в свободную среду промышленной разработки программ Eclipse, получено авторское свидетельство N2009610226 от 11.01.2009 о регистрации в реестре программ для ЭВМ, также выполнена реализация и для среды Microsoft Visual Studio.

Апробация. Основные положения диссертации докладывались на следующих конференциях, конкурсах и семинарах: конференция-конкурс ”Технологии Microsoft в теории и практике программирования” (Новосибирск, НГУ, 2007 и 2008 гг.); VI и VII Всероссийская научно-практическая конференция студентов, аспирантов и молодых ученых “Молодёжь и современные информационные технологии” (Томск, ТПУ, 2008 и 2009гг), V и VI Всероссийская научно-техническая конференция студентов, аспирантов и молодых ученых “Наука и молодёжь” (Барнаул, АлтГТУ, 2008 и 2009г.); Школа-семинар в Сибирском федеральном округе для участников (победителей) программы «Участник молодежного научно-инновационного конкурса» (Барнаул, АлтГТУ, 2008г.); Школа – семинар «Менеджмент технико-внедренческой деятельности» для победителей программы «У.М.Н.И.К.» 1 года (Томск, ТГУ, 2008); Конкурс IT проектов Алтайской торгово-промышленной палаты (Барнаул, 2008); VI и V Всероссийская научно-техническая конференция «Технологии Microsoft в теории и практике программирования» для студентов, аспирантов и молодых ученых Российской Федерации (Москва, МАИ, 2009, 2010); XII Региональная математическая конференция МАК-2009 (Барнаул, АлтГУ), также работа была представлена в сборниках конференций: XLVI Международная научно-студенческая конференция "Студент и научно-технический прогресс (Новосибирск, СО РАН, 2008), Всероссийская научно-техническая конференция студентов, аспирантов и молодых ученых “Научная сессия ТУСУР-2008” (Томск, ТУСУР, 2008); V Всероссийская конференция “Математическое моделирование и краевые задачи” (Самара, СамГТУ, 2008); X Международная Белорусская матема-

тическая конференция (Минск, БГУ, 2008).

В качестве педагогической практики результаты работы опробованы при проектировании распределенных программных систем студентами на лабораторных занятиях по курсу «Проектирование сетевых и многопоточных приложений».

Методы исследования включают методы математической логики, теории множеств, теории конечных автоматов, теории формальных систем, теории языков программирования и методов трансляции, темпоральной логики, объектно-ориентированный анализ, вычислительный эксперимент, эквивалентное преобразование моделей, технологии аспектно-ориентированного программирования и разработки, управляемой моделями.

Объем и структура работы. Диссертация состоит из введения, списка сокращений, трех глав, заключения и шести приложений. Список использованной литературы содержит 103 наименования, включая работы автора. Текст диссертации содержит 185 страниц машинописного текста, включая 46 рисунков, 2 таблицы.

На защиту выносятся:

1. Автоматная стохастическая мультиагентная модель, позволяющая моделировать поведение распределенных многокомпонентных программных систем на разных уровнях абстракции.
2. Процесс разработки и тестирования на основе построенной модели.
3. Алгоритмы динамического тестирования реальной программной системы по ее построенной модели.
4. Алгоритмы статического анализа свойств модели.

СОДЕРЖАНИЕ РАБОТЫ

1 Автоматная стохастическая мультиагентная модель, позволяющая моделировать поведение распределенных многокомпонентных программных систем на разных уровнях абстракции

В связи с наличием множества языков программирования и разнородных средств разработки ПО, целесообразным является создание адекватной математической модели, позволяющей описывать современные распределенные взаимодействующие системы с целью применения для тестирования.

Распределенной системой назовем систему, компоненты которой расположены на нескольких узлах сети или на одном узле с эмуляцией работы по сети.

Компонент распределенной системы— это отдельный программный «проект» в терминах интегрированной среды разработки (IDE), программа, которая имеет свою логику работы и может взаимодействовать с другим компонентами. Компонент может находиться на отдельном узле в распределенной системе и указываться при проектировании на UML диаграмме развертывания.

В предлагаемой математической модели программной системы можно выделить следующие уровни абстракции:

- на уровне всей взаимодействующей системы – структурный автомат;
- на уровне компонентов – комбинация расширенных вероятностных многопоточных конечных автоматов с обработкой событий и исключений, с возможностью моделирования межкомпонентных и внутрикомпонентных связей при помощи посылки сообщений, блокировки ресурсов и изменения функциональности посредством применения аспектов;
- на нижнем уровне – состояния, привязанные к реальному коду программного компонента, состояния могут группироваться в подавтоматы (например, логика работы потока задается подавтоматом) и рассматриваться как сверхсостояния.

Состояние s из множества состояний S относительно к коду компонента предлагается рассматривать как определяемую разработчиком последовательность $\{p, .., r\}$ линий кода Src_{fi} в исходном файле $f \in SrcFile$, которая, по его мнению, отражает неделимое состояние системы в каждый момент времени:

$$s = \{ \cup_{i=p...r} Src_{fi} | f \in SrcFile, \text{в файле } f \text{ с линии } p \text{ до линии } r \text{ код} - \text{логически значимый} \}.$$

В общем виде предлагаемая модель современной распределенной недетерминированной тестируемой системы имеет вид:

$$System = (Node, A^*, A_{structured}, BoundNodes, Msg, Res, CP(A^*), Asp).$$

Здесь:

$Node$ – множество узлов распределенной системы;

$BoundNodes$ - отображение, определяет расположение компонентов по узлам;

A^* – множество расширенных конечных автоматов, каждый из которых моделирует один компонент системы, для которой строится модель;

$A_{structured}$ – структурный автомат, модель высокого уровня системы;

Asp – множество аспектов, реализующих сквозную (общую) функциональность по принципам аспектно-ориентированного программирования. Аспект задается подавтоматом, точкой сопряжения для связи с подавтоматами, к котором он применяется, и способом применения (до, после, вместо). В работе доказывается теорема, что для автомата с аспектной группой существует эквивалентный автомат без аспектной группы с добавленными состояниями и переходами.

$CP(A^*)$ – множество точек сопряжения между моделями компонентов и для связи с аспектами по принципу рукопожатия;

Res – множество глобальных общих блокируемых ресурсов;

Msg – множество глобальных посылаемых сообщений.

Структурный автомат задается как четверка вида:

$$A_{structured} = (Cn, \delta_{structured}, InputPin, \Sigma).$$

Здесь:

Cn – состояния структурного автомата – компоненты моделируемой системы с учетом кратностей (кратность компонента — количество его одинаковых эк-

экземпляров, работающих в распределенной системе);

$\delta_{structured}$ – функция переходов структурного автомата. В структурном автомате переход между состояниями-компонентами системы означает отсылку сообщения между ними. Функция $\delta_{structured}$ определяет переход-отсылку сообщения в зависимости от компонента, номера сообщения, входа из множества $InputPin$, куда будет послано сообщение, и идентификатора сообщения из Σ.

Каждый элемент множества A^* представляет собой расширенный конечный автомат вида:

$$A=(S, s_0, \delta, \gamma, S_{fin}, E, Msg' \subseteq Msg, Res' \subseteq Res),$$

где Msg' и Res' - соответственно локальные для компонента распределённой системы подмножества глобальных множеств Msg и Res.

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

- Недетерминированной функцией переходов δ, она задает отображение:

$$\delta : S \times D \times P \times N \times T \times Msg' \times Res' \times \longrightarrow$$

$$2\text{^}(((T \times S)* \cup S) \times Msg'^* \times Res').$$

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 системы недетерминированно переходит или 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 некоторого ресурса из Res'.

- Функцией переходов «по ребрам» γ, определяющей возможность возникновения некоторого числа событий или исключительных ситуаций E при переходе из состояния в состояние $\gamma : S \times S \to E^*$.
- Операцией редукции, осуществляющий сворачивание потоков в один: $join : (S \times T)^+ \to S \times T$.

Следует обратить внимание на то, что все действия в расширенном автомате связаны с некоторой вероятностью, и здесь мы говорим прежде всего о моделировании потока управления (control flow), но не потока данных (data flow). Использование вероятностей позволяет моделировать недетерминированное сложное поведение, учет всех особенностей которого слишком сложен. Предлагаемая модель построена таким образом, что при ее создании и описании компонента системы в некотором состоянии мы можем выбирать, будем ли мы абстрагироваться от условий и делать вероятностные недетерминированные переходы или учитывать, например, пришедшее сообщение и делать детерминированный переход в зависимости от описываемых факторов.

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.

Апостериорными вероятностями тестируемой системы назовем вероятности переходов модели, вычисленные после работы программы на основании статистики переходов. Апостериорные вероятности определяют реальное поведение системы после её запуска.

Компоненты системы, каждый из которых задан расширенным автоматом A, могут взаимодействовать друг с другом через:

– посылку и получение сообщений из множества Msg;

– блокировку, ожидание раблокировки и разблокировку общих ресурсов из Res;

– точки сопряжения (взаимодействие типа «рукопожатие») из множества CP, задающие ожидаемую синхронизацию компонентов системы (пока один компонент находится в заданном наборе состояний, другой может находиться в другом заданном наборе состояний).

Поскольку описывать поведение сложными функциями перехода затруднительно, можно предложить альтернативный способ задания расширенного автомата как тройки из набора состояний, переходов и операций, которые могут произойти в данном состоянии (возможно, с генерацией событий):

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

где $Trans=S\to S\times E^*$ - отображение, определяет переход с возможной генерацией событий, $Op=\{fork, join, send, receive, block, unblock\}\times E^*$ - множество рассматриваемых ниже операций в расширенном автомате также с возможными связанными событиями. Под словом «*операция*» мы имеем ввиду логически обособленное действие по переходу из состояния расширенного автомата в его другое состояние с заданной вероятностью, выполняющее помимо перехода некоторые действия со множествами расширенного автомата. Операция является подмножеством функции перехода.

Вводятся следующие 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 выполнятся 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 $T_{slave}\subseteq T$. Обобщенно, $join: P\times(S\times T)^+\to S\times T$. 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 сообщения $send\subseteq(\delta\vee\gamma): S\times T\times P\to(S\times Msg)\vee E$.

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 сообщения $receive \subseteq (\delta \vee \gamma): S \times T \times P \times Msg \to S \vee E$.

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

$block \subseteq (\delta \vee \gamma): S \times T \times N \times P \times Res \to (S \times 1 \times Res) \vee E$. Находясь в состоянии из S в потоке из T с некоторой кратностью N с вероятностью P поток текущего компонента начинает блокировку ресурса из Res и при удачной блокировке переходит в новое состояние с кратностью равной 1. При неудачной блокировке поток зависает и ждет освобождения ресурса, при неуспешном ожидании может быть сгенерировано событие из E.

- Разблокировка разделяемого ресурса

$unblock \subseteq (\delta \vee \gamma): S \times 1 \times Res \times P \to (S \times N \times Res) \vee E$. Находясь в состоянии из S, владея заблокированным ресурсом Res и имея кратность состояния 1 (никакой другой поток не находится в данном состоянии) компонент с некоторой вероятностью P осуществляет попытку разблокировки ресурса Res, при успешной разблокировке переходит в новое состояние с любой допустимой кратностью, при неуспешной – может генерировать событие из E.

При проектировании рассмотренная модель системы может быть задана двумя видами диаграмм – диаграммой, описывающей модель верхнего уровня (структурный автомат), и связанными с ней диаграммами для каждой из моделей компонентов в виде расширенных конечных автоматов, а также дополнительной диаграммой, показывающей для распределенной системы расположение компонентов по узлам сети.

2 Процесс разработки и тестирования на основе предложенной модели

Рассмотрим применение тестирования на основе моделей в процессе разработки и основные используемые в процессе документы (рисунок 1). Первоначально разработка или итерация разработки начинается со сбора требований к системе. Здесь могут быть использованы спецификации и UML диаграммы, исходя из которых можно говорить о примерной структуре нашей модели для системы. После выяснения требований происходит разработка системы (написание кода, модульное тестирование, и одновременное описание модели). После завершения разработки необходимо провести тестирование реализованной функциональности по модели.

Тестирование на основе модели предлагается проводить двумя способами:

– Статическое тестирование (верификация и симуляция) по модели,

когда мы имеем модель системы и полностью заменяем систему на её модель. С реальным кодом системы, так и с выполняемыми компонентами, получившимися в результате компиляции исходного кода, работа не производится.

– Динамическое тестирование проводится на скомпилированной рабочей системе с целью проверки её поведения относительно построенной модели.

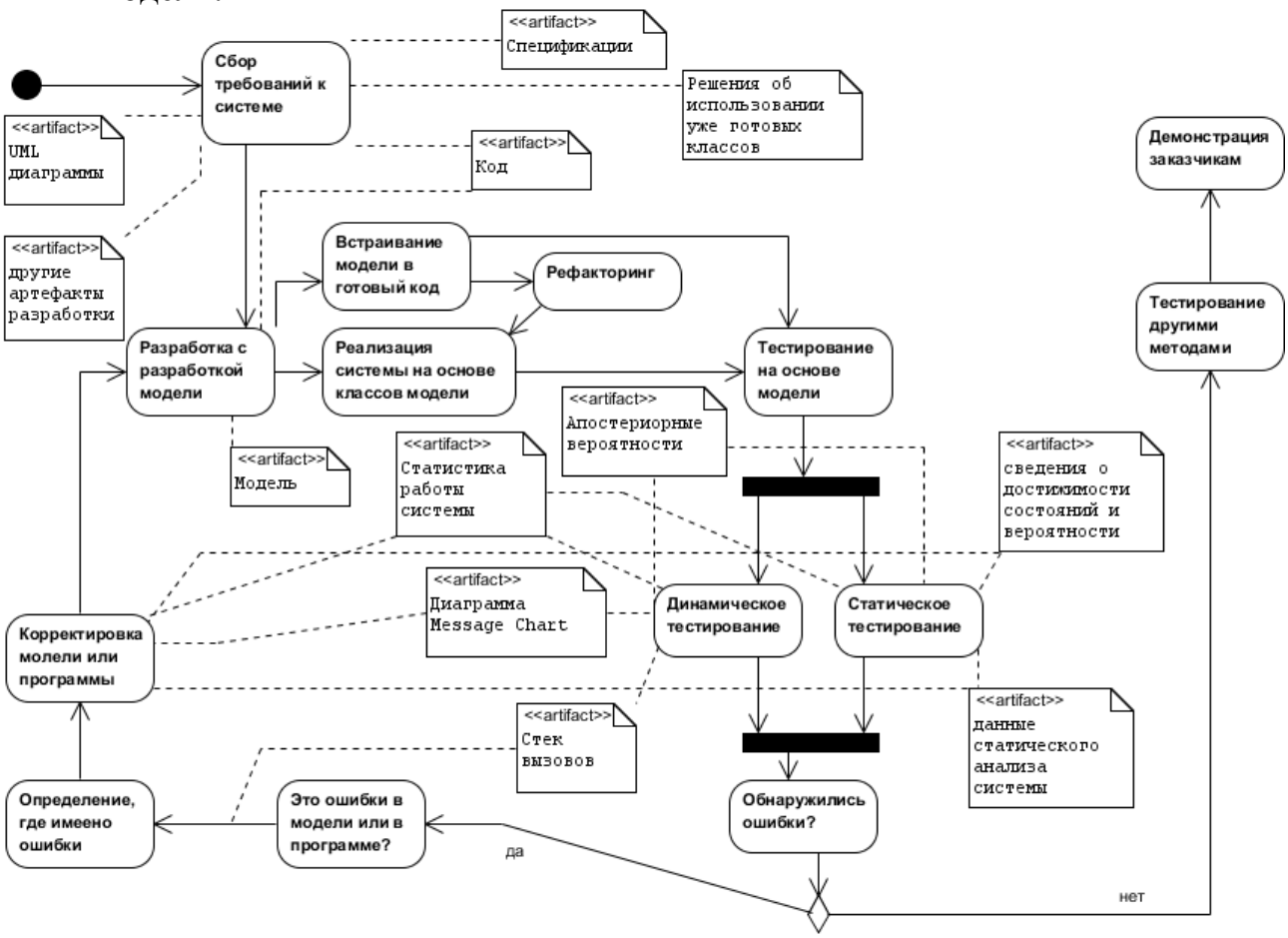

Рисунок 1 - Итерация процесса разработки

При обнаружении ошибки в процессе тестирования возможны варианты:

– допущена ошибка в программе, т.е. программа не соответствует описанной модели;

– несоответствующее поведение — это ошибка в описании модели, т.е. модель неправильно описывает систему.

После завершения проверки предлагаемыми в работе методами рекомендуется воспользоваться и другими, известными на сегодня методами и способами тестирования, поскольку нетрудно доказать путем сведения к неразрешимой проблеме переводимости или остановки машины Тьюринга (выкладки приведены в настоящей диссертации), что по имеющейся программе невозможно определить, содержит она ошибки или нет.

Описание модели. Пусть имеется распределенная многокомпонентная

система, которую необходимо описать и протестировать на основе модели. Возможны два варианта – система уже существует в виде программного кода и модель нужно построить по данному коду, либо система существует в форме спецификаций и диаграмм, описывающих ее поведение, и код для нее пока еще не написан. Модель должна строиться и в том, и в другом случае.

В данном исследовании как при отсутствии, так и при наличии кода системы предлагается строить модель по предлагаемому автором принципу - «код и модель — одно целое».

Для программных систем, код которых уже написан, необходимо встроить модель в исходный код. В работе предлагается описание модели на специальном языке описания совместно с исходным кодом. Переходы и состояния привязаны к файлам и строкам исходного кода, поэтому модель описывается на месте, где определяются состояния, переходы, сообщения, потоки и т.д. Описание модели в коде не вызывает ошибок компиляции, так как модель внедряется в псевдокомментариях к исходному коду, которые пропускаются при компиляции кода, но обрабатываются анализирующей и тестирующей системой. Таким образом, модель является метаданными к исходному коду.

Для описания модели разработан предметно-ориентированный язык: в результате объектно-ориентированного анализа математической модели создана модель «сущность-связь», которая описана с помощью формального языка с XML тегами. Для разработчиков реализовано расширение для среды разработки Eclipse, которое позволяет в редакторе среды выделять код, определять состояния, переходы и операции, генерировать описание модели в псевдокомментариях.

Для программных систем, находящихся в стадии проектирования, предлагается использовать программную реализацию модели в виде программного каркаса (системы классов), который определяет все взаимодействия с учетом состояний, переходов и операций.

Проектирование системы в этом случае осуществляется путем манипулирования разработанными классами (инстансцирования, задания их атрибутов, наследования, установки пользовательского кода, который выполняется в состояниях). При этом первоначально модель проектируется в виде диаграмм – структурного автомата, диаграммы распределения компонентов по узлам сети, расширенных многопоточных конечный автоматов для каждого из компонентов системы, далее автоматически генерируются конструкторы объектов-экземпляров классов мета-модели с параметрами, которые и определяют заданное пользователем поведение модели. После генерации пользователь может написать свой программный код (интерфейсы для этого определены), который определит поведение системы по выполнению нужных пользователю специфических действий, все взаимодействие в многокомпонентной распределенной системе при этом уже реализовано в предлагаемых классах. Процесс разработки полностью становится управляемым моделями (MDD).

Адекватность и практическая значимость модели. Для практического

использования предложенных методов разработки необходимо показать эквивалентность различных представлений модели, а также возможность её применения для описания широкого класса алгоритмов, поэтому в работе были сформулированы и доказаны следующие утверждения.

Утверждение 1. Представления автоматной стохастической мультиагентной модели системы в виде классов, на XML языке и её теоретико-множественное определение эквивалентны в описании взаимодействия.

Утверждение 2. С помощью автоматной стохастической мультагентной модели в виде классов можно реализовать любой параллельный алгоритм без использования внешних примитивов синхронизации и без внешнего управления потоком программы, на основе только переходов и операций стохастической мультиагентной модели.

Доказательство данного утверждения проводится путем описания возможности построения интерпретатора для формальной модели параллельного алгоритма, заданного машиной абстрактных состояний (ASM), использующего реализацию модели в виде классов и алгоритм LL(1) анализа.

Следствие. Для любого параллельного алгоритма можно построить автоматную стохастическую мультиагентную модель.

Утверждение 3. Любой распределенный алгоритм можно реализовать при помощи представления автоматной стохастической мультиагентной модели в виде классов.

Для доказательства сведем распределенный алгоритм к параллельному путем реализации интерпретаторов для распределенно работающих агентов, выполняющих шаги в машине ASM над синхронизированными общими данными при помощи блокировки и рассылки сообщений.

Следствие. Для любого распределенного алгоритма можно построить автоматную стохастическую мультиагентную модель.

3 Алгоритмы динамического тестирования реальной программной системы по ее построенной модели

Динамическое (on-line) тестирование предполагает построение динамической модели($M_{дин}$) работающей системы и сравнение ее с описанной моделью ($M_{стат}$), причем, если $M_{дин} \subseteq M_{стат}$, 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. Для обеспечения отсылки данных о проходящих событиях в

модели реализовано два метода:

а) если код системы был написан отдельно от модели, а модель была определена в комментариях к коду – выполняется автоматически сгенерированный код на целевом языке программирования в точках описания модели. Этот код взаимодействует с тестирующим сервером в процессе работы компонента, посылая все события на сервер, где они регистрируются в базе данных. Вставка дополнительного кода осуществляется препроцессором при сборке проекта.

б) если код системы реализован с использованием классов объектно-ориентированной модели, то вставка промежуточного кода не требуется, поскольку отсылку состояний на сервер в процессе работы системы способны передавать сами классы прозрачно для пользователя.

В процессе тестирования после разворачивания тестируемой системы по тестовым узлам (*BoundNodes*), запуска сервера, задания для него активной статической модели $M_{стат}$ и начала взаимодействия, на сервере начинается построение динамической модели системы $M_{дин}$ и ее сверка со статической.

Тестирование соответствия (conformance-testing) включает проверку следующих условий:

- Переход из заданного состояния в одно из заданных:

$$\forall\ A \in A^*: S(A) = S\ \&\ \forall\ s_1 \in S\ \&\ Next^1(s_1) \stackrel{\delta}{=} (S_{1\text{next}} \subseteq S) \Rightarrow s_1 \to s_2,\ s_2 \in S_{1\text{next}}.$$

Переходы из состояния в состояние производятся строго по модели. Переход в состояние, которое не соответствует функции переходов автомата, говорит об ошибке в потоке управления.

- Срабатывание событий и исключительных ситуаций.

При переходах:

$$\forall\ A \in A^*: S(A) = S\ \&\ \forall\ s_1 \in S\ \&\ Next^1(s_1) \stackrel{\gamma}{=} (S_{1\text{next}} \subseteq S) \Rightarrow s_1 \to s_2,\ s_2 \in S_{1\text{next}};$$

при отсылке сообщений:

$$M_{стат}: \exists\, msg \in Msg, \exists S_1 \in S, \exists S_2 \in s, send \subseteq (\delta \wedge \gamma): S_1 \times T \times P \to (S_2 \times msg) \vee E \Rightarrow$$
$$M_{дин}: S_1 \to S_2 \vee S_1 \to S_3, S_3 = first(A_E);$$

при блокировке ресурса:

$$M_{стат}: \exists\, S_1, S_2, block \subseteq (\delta \vee \gamma): S_1 \times T \times N \times P \times Res \to (S_2 \times 1 \times Res) \vee E \Rightarrow$$
$$M_{дин}: S_1 \to S_2 \vee S_1 \to S_3, S_3 = first(A_E).$$

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

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

$$\forall\ c_i \to c_j, c_i \in Cn, c_j \in Cn \Rightarrow \exists msg = (s_{\text{from}}, s_{\text{to}}, msgid, type, txt), s_{\text{from}} \in A_i,$$
$$s_{\text{to}} \in A_j, c_i = (A_i, n_i), c_j = (A_j, n_j).$$

Модель верхнего уровня ($A_{structured}$) описывает возможные взаимодействия через отправку/прием сообщений, модели компонентов более низкого уровня описывают обмен конкретных сообщений между заданными состояниями. Если

заданный порядок обмена нарушен, регистрируется ошибка взаимодействия.

- Корректность создания и завершения потоков, кратности состояний

$$\forall\ T_1 \dots T_k,\ fork: P \times S_{fork} \to (S_1 \times T_1) \times \dots \times (S_k \times T_k) =>$$
$$N(S_1) = N(T_1) = N(S_{fork}) + 1 \dots N(S_k) = N(T_k) = N(S_{fork}) + 1;$$
$$\forall\ T_1 \dots T_k,\ join: (S_1 \times T_1) \times \dots \times (S_k \times T_k) \to S_{join} \times T_{join} => N(S_{join}) = N(T_{join}) = 1$$

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

$$\forall\ msg, send: P \times S_{\text{from}} \times T_{send} \times P \to (S_{sent} \times msg) \vee E_{\text{not\_sent}},\ msg = (S_{\text{from}}, S_{\text{to}}, type, text)$$
$$\exists receive: P \times T_{receive} \times S_{\text{to}} \times msg \to S_{received} \vee E_{\text{not\_sent}}.$$

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

- Критические секции, связанные с блокировкой внешних ресурсов

$$\forall\ S, S_1 \geq S \geq S_{2,} S_1: block(res), S_2: unblock(res), res \in Res => N(S) = 1.$$

Если блокируется ресурс одним потоком /компонентом и производится попытка доступа к нему другим потоком/компонентом, то проверяется, что доступ к ресурсу имеет только его владелец. Кратность состояния в критической секции равна 1.

- Контроль возникающих блокировок – бесконечных ожиданий (deadlocks):

$$deadlock <=> \exists T_1, T_2, Host(R_2) = T_1, Host(R_1) = T_2, T_1 \neq T_{2,}$$
$$block: T_2 \times R_2 \,\&\, block: T_1 \times R_1.$$

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) не нарушается:

$$\forall\ CP(A_1, A_2) => \forall\ S_1 \in S(A_1) \exists S_2 \in S(A_2): \exists время\ t, S_t(A_1) = S_1 \,\&\, S_t(A_2) = S_2.$$

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

При обнаружении несоответствия поведения динамической и статической модели системы необходимо анализировать причины ошибки. Для этого, тестировщику-аналитику предоставляются следующие средства:

– Детальная трассировка посещенных состояний и примененных в процессе операций (StateTrace). В процессе тестирования запоминается информация о всех действиях в модели.

– Диаграмма последовательности обмена сообщениями. Она строится автоматически и отражает записанную тестирующим сервером последовательность обмена сообщениями между компонентами системы и их потоками.

– Статистика работы системы и апостериорные вероятности всех возникающих действий. Вычисляются реальные вероятности уже совершенных переходов и операций, которые могут быть использованы как исходные при статическом тестировании.

– Статистика по наиболее часто совершаемым переходам и использованию межкомпонентных связей и задержек при взаимодействиях (время перехода из состояния в состояние).

4 Алгоритмы статического анализа свойств модели

В данном исследовании используются методы Model checking применительно к задачам статического анализа свойств модели. Для их применения необходимо преобразовать разработанную модель в эквивалентное представление, которое можно обрабатывать алгоритмами верификации. С использованием нашего подхода пользователю-тестировщику при работе с моделью с помощью разработанных средств не нужно знать, какие модели используются внутри реализации для проведения тестирования.

В работе была сформулирована и доказана методом построения **теорема:** для автоматной стохастической мультиагентной модели существует эквивалентная модель, описываемая на языке Promela для верификатора Spin.

Рассмотрим теперь, как задачи статического тестирования могут быть решены при помощи полученной эквивалентной модели и правил темпоральной логики.

- Симуляция по модели.

Сначала автоматная стохастическая мультиагентная модель преобразуется в процессы Promela, которые действуют исходя из недетерминированного выбора следующего состояния, после чего можно проводить симуляцию по модели с использованием средств и режимов верификатора Spin:

а) случайная прогулка производится путем случайного выбора траектории поведения модели. Случайная прогулка осуществляется с учетом вероятностей переходов и операций;

б) отладка по модели производится запуском верификатора в интерактивном режиме, чтением его вывода, отображением с помощью графических средств взаимодействия с пользователем вариантов продолжения на шаге, требующим недетерминированного выбора перехода или операции, и записью варианта в поток ввода верификатора.

Такие действия, как анализ вероятностей, проверка достижимости, связности, правильности приема и передачи сообщений и доступа к ресурсам, предполагают генерацию специальных переменных в модели на Promela, участвующих в проверяемых предикатах темпоральной логики.

- Анализ вероятностей.

Вероятности могут быть как определены по модели, так и выгружены с сервера динамического тестирования после запуска реальной системы на выполнение и использованы для воспроизведения сценария работы системы на модели.

а) Для подсчета вероятности достижения заданного состояния $s_x \in S$ из начального, прежде всего, нужно обеспечить, чтобы одна из случайных траекторий работы системы проходила через нужное состояние s_x. Это можно реализовать при помощи задания предиката, значение которого предполагается истинным во всех состояниях: $state \neq s_x$ (здесь $state$ — глобальная переменная, в которой сохраняется текущее состояние, для каждого главного потока компонента и любого побочного потока она своя; в предикате используется переменная именно для того компонента и потока, в котором находится состояние s_x), и запуска верификатора в режиме доказательства правильности, при этом данный предикат становится ложным при очередном выборе траектории с достижением состояния s_x и верификатор возвращает данную траекторию как контр-пример. Далее, с этой траекторией, содержащей заход в состояние s_x, запускается верификатор уже в режиме симуляции. Модель для верификатора генерируется таким образом, чтобы после каждого перехода или операции производился пересчет условной вероятности в каждый текущий момент. При достижении состояния s_x выводятся накопленные вероятности текущего потока и всех его родительских, эти данные читаются из потока вывода верификатора и становятся доступными как результат.

б) Для подсчета вероятности достижения заданного состояния $s_x \in S$ из другого заданного $s_y \in S$ необходимо действовать аналогично а), только необходимо установить траекторию, проходящую через оба состояния. Предикатом, который должен быть опровергнут, является $(state \neq s_y)\, U\, (G\, state \neq s_x)$, где U — темпоральный оператор «до тех пор, пока», а G — темпоральный оператор «всегда». Сама вероятность вычисляется как частное накапливаемых вероятностей для s_x и s_y.

- Проверка достижимости всех состояний.

Такая проверка осуществляется для каждого состояния $s_i \in S$, путем ожидания контр-примера для предиката ($state\ != s_i$). Если верификатор возвращает ошибку, то состояние s_i достигается, при этом как контр-пример можно найти путь из начального состояния в s_i.

- Проверка связности через точки сопряжения.

Связность через точки сопряжения – ожидаемый пользователем набор соответствий подавтомата (набора состояний) одного компонента или потока с подавтоматом (состояниями) другого. Пусть имеется точка сопряжения, связывающая некоторые логические наборы состояний A и B, $cp \in CP : \{ s_{A1}, s_{A2}, \ldots s_{An} \} \leftrightarrow \{ s_{B1}, s_{B2}, \ldots s_{Bm} \}$. Тогда проверка связности представляет собой проверку предиката

G (($state_A == s_{A1}$ || $state_A == s_{A2}$ ||.. $state_A == s_{An}$)->($state_B == s_{B1}$ || $state_B == s_{B2}$ ||.. $state_B == s_{Bm}$)).

Здесь $state_A$ и $state_B$ — те состояния, которые идентифицируют либо текущее состояние потока *main* компонента, где находится группа состояний A или B соответственно, либо одного из побочных потоков, где эти группы состояний могут быть объявлены. При ошибке верификации выдается контр-пример,

показывающий, как можно нарушить неправильно описанную связность между точками сопряжения.

- Проверка, что для каждое отправленное сообщение получено.

Для каждого сообщения генерируется своя переменная $status\_message_i$, которая может содержать два значения из перечисляемого типа *sent* (сообщение отправлено) и *received* (сообщение получено), изменение этих значений вставляется соответственно после отправки и после получения сообщения из канала. Предикат, истинность которого проверяется – $G((status\_message_i == sent) \rightarrow F(status\_message_i == received))$, где G и F — темпоральные операторы соответственно «всегда» и «хотя-бы раз в будущем».

- Проверка, что каждый заблокированный ресурс разблокируется.

Ресурсы res_i задаются переменными, принимающими значение 1 (ресурс заблокирован) или 0 (ресурс разблокирован), предикат для проверки условия будет иметь вид $G((res_i == 1) \rightarrow F(res_i == 0))$.

Средствами, которые могут помочь пользователю в процессе обнаружения ошибки и понять ее причины, получаемые в процессе статического тестирования, являются диаграмма сообщений и стек вызовов.

– Диаграмма сообщений (message sequence chart) — стандартная диаграмма языка UML, показывает изменение состояний процессов с течением времени и их связи через сообщения.

– Стек вызовов содержит все состояния, переходы и операции в том порядке, в котором они встречались в результате симуляции по модели.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ РАБОТЫ

Основные результаты работы заключаются в следующем:

1. Проведен анализ предметной области, моделей распределенных программных систем и средств для тестирования по модели.
2. Разработана, исследована, обоснована теоретически, практически и экспериментально модель современных распределенных программных систем на разных уровнях абстракции на основе комбинаций расширенных вероятностных многопоточных конечных автоматов, предложена графическая нотация для описания модели.
3. Разработаны методы описания модели как дополнения к коду, так и в качестве каркаса для разработки программных систем. Модель может быть описана в виде экземпляров классов на языке Java и в виде XML тегов.
4. Разработан и реализован в виде прототипов (подключаемых модулей к промышленным средам разработки Eclipse и Visual Studio) комплекс программ для описания модели и проведения тестирования.
5. Разработаны алгоритмы и инструментальные средства для статического и динамического тестирования по предложенной модели.

Рассмотренный подход позволяет практически применить разработку и тестирование по модели в промышленных процессах создания ПО.

РАБОТЫ АВТОРА ПО ТЕМЕ ДИССЕРТАЦИИ

Работы, опубликованные в журналах из перечня ведущих рецензируемых журналов ВАК Министерства образования и науки РФ:

1. Старолетов С.М. Проведение on-line тестирования программного обеспечения на основе построенной модели / Старолетов С.М., Крючкова Е.Н. // Ползуновский вестник № 2, 2010. – Барнаул: изд-во АлтГТУ, 2010. – с. 212-216.
2. Старолетов С.М. Динамическое тестирование распределенных систем на основе автоматных моделей. // Крючкова Е.Н., Старолетов С.М. Программная инженерия № 2, 2011. – М.: изд-во «Новые технологии», 2011. с. 22-27.

Другие работы по теме диссертации:

3. Старолетов С. М. Моделирование распределенных многокомпонентных программных систем и их тестирование на основе автоматных вероятностных моделей / Монография. С. М. Старолетов, Е. Н. Крючкова. – Барнаул : Изд-во АлтГТУ, 2011. – 107 с. – ISBN 978-5-7568-0859-9
4. Staroletov S. 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. – N., 2011. p. 33-38
5. Старолетов С.М. Тестирование распределенных приложений на основе построения моделей / Е.Н. Крючкова, С.М. Старолетов // Прикладная информатика. – N 6(18), 2008, – М.: Market DS Publishing, с. 124-134.
6. Старолетов С.М. Применение моделей при тестировании программ / Е.Н. Крючкова, С.М. Старолетов // Ползуновский альманах. – Барнаул, 2008.- № 4.- с.16-21.
7. Старолетов С.М. Анализ моделирования оптимальных стратегий тестирования современных распределенных недетерминированных систем / Е.Н. Крючкова, С.М. Старолетов. Технологии Microsoft в теории и практике программировании. Конференция-конкурс работ студентов, аспирантов и молодых учёных. Тезисы докладов. – Н.: Оригинал 2, 2007. – с. 37-38.
8. Старолетов С.М. Конечный автомат с вероятностными переходами как модель распределённой программной системы/Е.Н. Крючкова, С.М. Старолетов. Математическое моделирование и краевые задачи: Труды пятой Всероссийской научной конференции с международным участием Ч.4.: Информационные технологии в математическом моделировании.- Самара: СамГТУ, 2008. – с. 129.
9. Старолетов С.М. Математическая модель для тестирования программ/Е.Н. Крючкова, С.М. Старолетов. VI Всероссийская научно-техническая конференция студентов, аспирантови молодых ученых "Наука и молодежь − 2009". Секция «Информационные иобразовательные технологии». Подсекция «Программное обеспечение вычис-лительной техники и автоматизированных систем». / Алт. гос. техн. ун-т им.И.И.Ползунова. – Барнаул: изд-во АлтГТУ, 2009. – с. 82-92.
10. Старолетов С.М. Методология тестирования программных систем на основе построения и анализа модели программы / Е.Н. Крючкова, С.М. Старолетов. Сборник трудов VII Всероссийской научно-практической конференции студентов, аспирантов и молодых ученых «Молодежь и современные информационные технологии». Томск, 25 - 27 февраля 2009 г., ч.1. Томск: Изд-во СПБ Графикс, 2009. – с. 195-196.
11. Старолетов С.М. Моделирование связной функциональности многокомпонентных систем / Старолетов С.М., Крючкова Е.Н. Тр. VII Всерос. конф. студентов, аспирантов и молодых учёных Центральный регион. Москва, 21-22 апреля 2010г. – М.: Вузовская книга, 2010. – с. 62-63
12. Старолетов С.М. Модель для тестирования ПО / С.М. Старолетов. Материалы двенадцатой конференции по математике "МАК-2009". – Барнаул: Изд-во Алт. Ун-та, 2009.

– с. 84-86.
13. Старолетов С.М. Мультиагентная модель распределенной системы для проведения model-based testing современного ПО / С.М. Старолетов, Е.Н. Крючкова. Тр. VI Всерос. конф. студентов, аспирантов и молодых учёных. Центральный регион. Москва, 1-2 апреля 2009г. – М.: Вузовская книга, 2009. – с. 37.
14. Старолетов С.М. Некоторые аспекты тестирования распределенных систем с построением модели / Е.Н. Крючкова, С.М. Старолетов. Научная сессия ТУСУР-2008: Материалы докладов Всероссийской научно-технической конференции студентов, аспирантов и молодых ученых. Томск, 5-8 мая 2008 г.: В пяти частях. Ч.2. – Томск: В-Спектр, 2008. – с. 50-53.
15. Старолетов С.М. Разработка программного комплекса для тестирования недетерминированных распределённых систем с построением моделей / Е.Н. Крючкова, С.М. Старолетов // Ползуновский альманах. – Барнаул: изд-во АлтГТУ, 2008. – № 4. – с. 219-220.
16. Старолетов С.М. Разработка системы тестирования распределенных приложений на основе математических моделей / С.М. Старолетов, Е.Н. Крючкова. Технологии Microsoft в теории и практике программирования. Конференция-конкурс работ студентов, аспирантов и молодых учёных. Тезисы докладов. – Н.:Оригинал 2, 2008. – 2 с.
17. Старолетов С.М. Реализация системы тестирования распределенных и многопоточных приложений на основе технологии МВТ / С.М. Старолетов, Е.Н. Крючкова. Измерение, контроль, информатизация. Международная научно-техническая конференция. Тезисы докладов. – Барнаул: изд-во АлтГТУ, 2011. – с. 16-20
18. Старолетов С.М. Свой взгляд на Model Based Testing распределенных недетерминированнных систем / Старолетов С.М., Крючкова Е.Н. Сборник трудов региональной конференции-конкурса (Северо-Западный регион) студентов, аспирантов и молодых ученых «Технологии Microsoft в теории и практике программирования 2009»Санкт-Петербург, 17-18 марта 2009. – Изд-во Политехн. ун-та, 2009. - с. 173.
19. Старолетов С.М. Тестирование недетерминированного программного обеспечение на основе моделей / С.М. Старолетов. Тезисы докладов Международной научной конференции "X Белорусская математическая конференция". Минск, 3-7 ноября 2008 года. – с. 90.
20. Старолетов С.М. Тестирование распределенных программ с построением модели программы в виде конечного автомата с вероятностными переходами / Е.Н. Крючкова, С.М. Старолетов. Материалы XLVI Международной научной студенческой конференции «Студент и научно-технический прогресс»:Информационные технологии. – Новосиб. гос. ун-т. Новосибирск, 2008. – 2с.
21. Старолетов С.М. Тестирование распределенных систем с помощью построения моделей / Е.Н. Крючкова, С.М. Старолетов. Молодежь и современные информационные технологии. Сборник трудов VI Всероссийской научно-практической конференции студентов, аспирантов и молодых ученых. Томск, 26 - 28 февраля 2008 г. – с. 159-161

Подписано в печать 15.11.2011
Печать – цифровая. Усл.п.л. 1,16.
Тираж 120 экз. Заказ 2011 –
Отпечатано в типографии АлтГТУ,
656038, г. Барнаул, пр-т Ленина, 46
тел.: (8-3852) 36-84-61
Лицензия на полиграфическую деятельность
ПЛД №28-35 от 15.07.97 г.