Обзор подходов к верификации распределенных систем

И.Б. Бурдонов, А.С. Косачев, В.Н. Пономаренко, В.З. Шнитман.

1. Введение

1.1. Введение в распределенные системы

Распределенными системами называются программно-аппаратные системы, в которых исполнение операций (действий, вычислений), необходимых для обеспечения целевой функциональности системы, распределено (физически или логически) между разными исполнителями. В каждом исполнителе поддерживается свой поток управления исполнением операций. Традиционно рассматривают два вида распределенных систем – параллельные системы и собственно распределенные системы.

К параллельным системам традиционно относят программные системы, в которых потоки управления используют общее пространство адресной памяти.

Собственно распределенные системы, как правило, включают подсистемы, потоки управления которых обладают собственным адресным пространством, недоступным для потоков управления из других подсистем. В ещё более узком понимании распределенная система содержит подсистемы, которые *физически* распределены в пространстве, то есть функционируют на разнесенных в пространстве компьютерах.

В последнее время граница между распределенными и параллельными системами становится все более расплывчатой. По этой причине в нашем обзоре мы не разделяем эти понятия.

Под потоком управления мы понимаем последовательное исполнение инструкций программы. При таком рассмотрении мы отвлекаемся от деталей выполнения инструкции в процессоре и сосредотачиваем свое внимание на логике управления данными. По этой причине в обзор не вошли методы, используемые при верификации средств *физического* распараллеливания вычислений в аппаратуре.

Формальное определение *одновременности* выполнения потоков управления дать нелегко. Мы ограничимся полуформальным представлением. Скажем, что два потока управления выполняются одновременно, если в течение

небольшого промежутка времени могут выполняться программные инструкции обоих потоков.

Пожалуй, важнейшим аспектом распределенных систем является *взаимодействие* между частями системы, так как именно взаимодействие порождает основные трудности как при разработке, так и при верификации распределенных систем.

Примеры взаимодействий внутри распределенной системы. В зависимости от архитектуры распределенной системы, от способов её декомпозиции для организации взаимодействий используются различные методы. Например, взаимодействие через разделяемую память (при этом требуется дополнительное взаимодействие по синхронизации операций над разделяемой памятью), взаимодействие путем вызовов удаленных процедур, путем организации транзакций (клиент-серверное взаимодействие), взаимодействие путем обмена сообщениями. Важным аспектом функционирования распределенной системы является доступ частей системы к разделяемым ресурсам. При параллельном выполнении потоков управления может сложится ситуация, в которой к одному ресурсу одновременно обращаются несколько потоков управления. Здесь одновременность означает, что за время, необходимое для выполнения логического запроса некоторого потока управления к ресурсу, могут поступить запросы от других потоков к тому же самому ресурсу. Важно, что запрос логический, так как разделение одновременных физических запросов к аппаратуре ресурса мы не рассматриваем.

Примеры. В многопотоковом приложении можно рассматривать оперативную память как разделяемый ресурс, тогда операция изменения значения некоторой переменной дает пример логического запроса к ресурсу. В системах управления базами данных ресурсами можно назвать таблицы базы данных, а логическими запросами – запросы на чтение/изменение данных в таблицах.

Но при всей «распределенности» система извне, с точки зрения её окружения остается единым целым, так почему создаются распределенные системы? В чем их преимущества перед централизованными ЭВМ?

- 1-ая причина – экономическая. Закон Гроша (Herb Grosh, 25 лет назад) – быстродействие процессора пропорциональна квадрату его стоимости. С появлением микропроцессоров закон перестал действовать – за двойную цену можно получить тот же процессор с несколько большей частотой.
- 2-ая причина – можно достичь такой высокой производительности путем объединения микропроцессоров, которая недостижима в централизованном компьютере.
- 3-я причина – естественная распределенность (банк, поддержка совместной работы группы пользователей).
- 4-ая причина – надежность (выход из строя нескольких узлов незначительно снизит производительность).

– 5-я причина – наращиваемость производительности. В будущем главной причиной будет наличие огромного количества персональных компьютеров и необходимость совместной работы без ощущения неудобства от географического и физического распределения людей, данных и машин.

Последнее десятилетие характеризуется возникновением задач в науке, технике и бизнесе, требующих сверхвысоких вычислительных ресурсов и производительности сетей. Ответом индустрии было появление все более мощных компьютеров и сложного ПО. Тем не менее в этих областях все еще существует ряд задач, решение которых не под силу даже современным суперкомпьютерам. Как правило, проблемы здесь связаны с интенсивными вычислениями или/и обработкой громадных массивов данных. Их решение требует использования разнородных ресурсов, тогда как один суперкомпьютер не в состоянии их предоставить. Специалисты видят выход в создании сетей вычислительных ресурсов, или Grid Computing.

Новый подход имеет несколько наименований: метавычисления (metacomputing), непрерывно масштабируемые вычисления (seamless scalable computing), глобальные вычисления (global computing). Однако в последнее время большее признание для обозначения новой технологии получило выражение Grid Computing, которое и будет использоваться в дальнейшем без перевода ввиду отсутствия подходящего русскоязычного эквивалента.

Идея, лежащая в основании Grid Computing, состоит в предоставлении вычислительных ресурсов и устройств массовой памяти таким же способом, каким поставляется электроэнергия с помощью единой энергосистемы. Это достигается посредством сложного механизма кластеризации ресурсов в Internet.

Создание столь широкомасштабной инфраструктуры требует определения и принятия стандартных протоколов и сервисов, аналогичных TCP/IP, являющихся сердцем Internet. Обычного процесса разработки и принятия необходимых стандартов для этой технологии сегодня не существует, хотя Grid Forum ведет работы в данном направлении. В то же время большинство существующих Grid-проектов построены на протоколах и сервисах Global Toolkit, разработанных группой, возглавляемой Яном Фостером (Ian Foster), при Аргоннской Национальной Лаборатории, которые и послужили основным источником при подготовке этой публикации.

Концепция Grid Computing возникла в середине 90-х, когда для решения сложных научных и технических задач был реализован проект объединения 17 географически удаленных суперкомпьютеров в Северной Америке. С тех пор в построении подобных проектов достигнут не только значительный прогресс, но и появилась новая технология, существенно отличающаяся от современных основных технологий распределенных вычислений, которые не обеспечивают решения возникших проблем и не удовлетворяют требования координированного и динамического распределения ресурсов. Так, например, современные Internet-технологии направлены на коммуникации и обмен

информацией между компьютерами, но не обеспечивают интегрированного подхода к координированному использованию ресурсов на множестве сайтов для выполнения вычислений. Технология business-to-business сосредоточивается на разделении информации (часто с помощью централизованных серверов). Технологии корпоративных распределенных вычислений, такие, как CORBA и Enterprise Java, позволяют разделять ресурсы, но только в пределах одной организации. Возможности и ресурсы, предоставляемые провайдерами услуг по аренде приложений (Application Service Provider) и дискового пространства (Storage Service Provider), весьма ограничены. Пожалуй, единственной технологией, поддерживающей разделение ресурсов разных сайтов, является Distributed Computing Environment (DCE), но для виртуальных организаций она негибка и слишком обременительна. Радикально изменить картину могут только Grid-технологии. Какие же основные проблемы они должны решить?

Прежде всего, это гетерогенность. Технология Grid Computing подразумевает взаимодействие множества ресурсов, гетерогенных по своей природе и расположенных в многочисленных и географически удаленных административных доменах. Далее, это расширяемость. В пул объединяемых ресурсов может входить от нескольких элементов до нескольких тысяч и более. При этом возникает потенциальная возможность снижения производительности по мере увеличения пула. Следовательно, приложения, которые требуют для своего решения объединения большого числа географически удаленных ресурсов, должны разрабатываться таким образом, чтобы быть минимально чувствительными к времени задержки. И наконец, обеспечение динамичности и адаптивности. Дело в том, что при объединении большого количества ресурсов отказы элементов являются не исключением, а правилом. Поэтому управление ресурсами или приложениями должно осуществляться динамически, чтобы извлечь максимум производительности из доступных в данное время ресурсов и сервисов.

Последние пять лет усилий в этом направлении увенчались разработкой протоколов, сервисов и инструментов, позволяющих создавать среду разделения ресурсов, обладающую необходимыми свойствами.

1.2. Введение в задачу верификации

1.2.1. Проверка поведения систем

Верификация системы в самом общем смысле – это проверка соответствия между требованиями к системе и свойствами работающей системы. Существует три возможных подхода к верификации системы – тестирование, имитация (simulation) и математический подход, называемый формальной верификацией. Последний становится все более актуальным в связи с трудностями, возникающими для первых двух подходов, и в связи с бурным развитием последнего.

Современные распределенные системы все больше опираются в своей деятельности на ПО, которое не только увеличивается значительно в своих

размерах, на также все больше зависит от таких характеристик, как параллельность и распределенность. Для распределенных систем реального времени актуальными являются такие ее свойства, как структура системы и управление временем.

Традиционное тестирование таких систем, ориентированное на разработку тестовых наборов, состоящих из входных воздействий и ожидаемых реакций, становится все более ненадежным, т.к. в распределенных системах реального времени проблематично воспроизвести ситуацию, приводящую именно к таким реакциям. Как в сфере научных исследований, так и разработке промышленного инструментария для верификации распределенных систем, заметна тенденция к концентрации усилий к продвижению формальных верификаций для параллельных и распределенных систем, которые обеспечивают гарантию того, что системы будут соблюдать свои ключевые свойства для всех своих исполнений.

Итак, формальная верификация связана с систематическим, математическим подходом, который включает

- модель системы, которая обычно представляет формально
  - состояние системы,
  - переход из одного состояния в другое,
- метод спецификации для представления желаемых свойств на некотором логическом языке,
- метод доказательства того, что модель удовлетворяет этим свойствам.

В распределенных системах особую актуальность приобретает описание поведения системы с помощью модели, что называется моделированием поведения. Моделирование поведения является основой для систематических подходов играет важную роль для формальной записи требований к системе, для спецификации, проектирования, симуляции, генерации кода, тестирования и верификации. Моделирование поведения позволяет создавать нотации, методы и инструменты для всех перечисленных стадий разработки ПО для распределенных систем.

1.2.2. Аспекты распределенной системы, подлежащие проверке

Качество ПО определяется как набор возможностей и характеристик продукта, который проявляет свою способность удовлетворять указанным требованиям (например, соответствовать спецификациям). Качество можно также рассматривать как уровень совокупности желаемых свойств, которым обладает программный продукт, или как степень выполнения ожиданий заказчика или пользователя от свойств данного продукта.

В стандарте ANSI/IEEE качество ПО понимается как набор характеристик продукта, который определяет степень, с которой ПО будет удовлетворять ожидания заказчика [ANSI/IEEE Standard 729-1983].

Качество ПО характеризуется такими свойствами, как корректность, надежность, возможность сопровождения и переносимость. Качество ПО включает также его пригодность для целевого назначения, разумную стоимость, простоту использования и характеристики апгрейда.

Понятие качества распределенной системы включает в себя **функциональные и нефункциональные требования**. Под функциональными требованиями понимаются свойства целевого назначения системы, а к нефункциональным свойствам часто относят такие характеристики, как надежность, безопасность, свойства системы реального времени, т.е. выполнение временных ограничений или своевременность.

Распределенным системам присущи некоторые нефункциональные требования на качество сервиса, обеспечиваемого системой. Такими нефункциональными требованиями являются **производительность, безопасность** [Dai03]**, надежность**.

Надежность является важным качеством распределенных систем, т.к. сбой может привести как к легко исправляемой ошибке, так и к катастрофическим потерям, В связи с этим распределенная система проектируется так, чтобы она была устойчивой к ошибкам.

Обеспечение качества (SQA – software quality assurance) системы понимается как планируемый и систематический подход к выполнению всех деятельностей, обеспечивающих адекватную уверенность в том, что продукт соответствует установленным техническим требованиям.

В стандарте IEEE управление качеством определяется как набор активностей, призванных установить пригодность к использованию процесса разработки продукта [IEEE Std 610.12-1990].

Важными процессами для подтверждения желаемого качества системы являются **верификация и валидация (V&V – verification and validation)**. В стандарте IEEE совместное определение этих двух процессов звучит так: верификация и валидация – это процесс установления, являются ли требования для системы или ее компонента полными и корректными, удовлетворяют ли продукты каждой стадии разработки требованиям или условиям, выдвинутыми для них на предыдущих стадиях, и соответствует ли конечная система или ее компонент специфицированным требованиям [IEEE Std 610.12-1990].

Анализ качества системы состоит из проверки моделей, моделей аналитического решения, или имитации моделей системы, которые описываются с помощью формализмов, таких как CSP, CCS, цепочки Маркова, сети Петри и т.д. Однако разработчики систем обычно предпочитают использование языков описания архитектуры и объектно-ориентированных нотаций для создания моделей систем и оценки качества системы.

Формальная верификация означает использование методов математической аргументации для доказательства корректности какой-либо системы.

Легко видеть бросающиеся в глаза возражения против применения формальной верификации:

- доказательства строятся для модели, а не реальной системы,
- спецификации страдают ошибками, неполнотой,
- ПО, созданное или протестированное с помощью теоретических выводов, возможно, будет ненадежным,
- доказательства, сделанные с помощью компьютера, являются нечитабельными.

Однако эти возражения можно обойти, если конечной целью является не математическая определенность, а скорее *методология*, следуя которой можно создавать более надежные программы и/или системы.

На самом деле, существует реальная большая проблема на этом пути, состоящая в том, что программы и системы являются сущностями намного более сложными, чем теоремы в математике. Вопрос применения математического доказательства к таким громоздким объектам решается с помощью двух вещей:

- автоматизации с помощью компьютера,
- композиционного подхода – разбиение больших объектов на более мелкие и доказательства свойств этих мелких объектов.

Почти все практические исследования в области формальной верификации направлены на поиски подходов к решению одной из этих проблем.

Автоматизированная валидация распределенных систем остается желаемой целью для сообщества индустриального производства корректных систем, таких как коммуникационные протоколы или встроенные системы. Несмотря на многочисленные выдающиеся научные исследования и разработки инструментов, сделанные в этой области, эта деятельность остается крайне трудной на практике: с одной стороны первоначальные описания ПО обычно делаются с использованием высокоуровневого формализма (либо языка программирования, либо формальной нотации как Lotos [ISO88], SDL [ITU99] or UML [OMG99]), и, с другой стороны, – многочисленные инструменты, пытающиеся покрыть процесс разработки и оперирующие на различных уровнях описания программ.

1.2.3. Тестирование

Тестирование всегда было существенной активностью для валидации корректности программных и аппаратных систем. Хотя тестирование не может гарантировать соблюдение свойств системы, как это возможно при формальной верификации, тестирование значительно увеличивает эффективность верификации системы.

Тестирование является трудным и дорогим процессом, а при тестировании распределенных систем встают такие вопросы, как интероперабельность, синхронизация, управление временем и параллельность. Кроме того, ошибки, обычно воспроизводимые в последовательных программах, не могут быть репродуцированы в параллельных и распределенных программах.

Существенную роль в автоматизации тестирования играют формальные подходы, которые поддерживают спецификацию и верификацию систем с помощью методов и техник, основанных на математических и логических подходах. Эти методы обеспечивают разработчиков систем возможностью анализировать системы и делать выводы о них с математической точностью и строгостью. Использование формальных методов не ограничивается ранними стадиями процесса разработки, но успешно применяется на разных этапах тестирования. Тестирование с применением формальных подходов доказывает также, что имеет смысл использовать формальные методы в процессе разработки ПО.

1.2.4. Мониторинг

Для достижения высокой производительности в распределенных системах требуется огромное количество информации о системе, о программах, выполняющихся в системе, и о специфике выполнения программ.

Мониторинг является мощным средством получения информации о компонентах распределенной системы для контроля над их поведением и принятия управленческих решений. Мониторинг также используется для получения информации об исполнении компонент и их взаимодействии при отладке распределенных и параллельных систем.

В распределенных системах часто используются системы мониторинга, которые собирают, анализируют информацию и дают возможность управлять производительностью системы, позволяют повысить надежность систем. В таких системах компоненты системы предоставляют два интерфейса – функциональный и служебный интерфейс, который используется для получения информации о состоянии компонентов и передачи управляющих воздействий.

1.2.5. Анализ производительности

Анализ производительности включает построение вероятностной модели рассматриваемой системы (обычно Марковский процесс), и затем выполнение ряда вычислительных задач, чтобы подсчитать статистические вероятности и связанные с ними оценки производительности (использование ресурсов, среднее время ожидания вызова и т.д.) [Wel02].

1.2.6. Критерии корректности

Верификация системы заключается в проверке правильности функционирования системы, которая определяется в соответствии с неким набором правил, которые называются критериями корректности. Критерий корректности, в соответствии с которым система верифицируется, задается в виде некой системной спецификации. Эта спецификация предписывает, что система должна делать, а что не должна и, следовательно, устанавливает базис для любой верификационной деятельности.

1.3. Цели исследования

В конечном итоге результаты исследования позволят решать вопросы повышения надежности, отказоустойчивости, соответствия стандартам распределенных систем (РС) разных видов. В методологическом плане исследование нацелено на систематизацию различных аспектов функционирования распределенных систем, на асинхронность, надежность, защищенность. При этом будут исследованы методы спецификации отдельных аспектов и методы верификации, которые с одной стороны, позволяют фокусироваться на отдельных аспектах, и, с другой стороны, рассматривать их в совокупности.

Среди подходов к анализу РС рассмотрены формальные модели, имитационное моделирование (simulating), измерения реальных систем.

Отправной точкой для любого метода верификации является выбор формализма (мета-модели), который будет использоваться для построения моделей. Рассмотрены следующие подходы: исполняемые модели, ограничительные модели (пропозициональные логики и исчисления предикатов, временные логики), аксиоматические модели (алгебраические и коалгебраические теории, исчисления параллельных процессов), а так же ряд иных подходов.

Из методов верификации, которые ограничиваются задачей установления корректности моделей РС, рассмотрены методы проверки модели (model checking), включая символьную (статическую) и динамическую проверку моделей, и вероятностную проверку моделей.

Проверка моделей дает ответ на вопрос, корректной ли является модель, на основе которой строиться РС. Для того, чтобы ответить на вопрос о корректности собственно РС, необходим статический и/или динамический анализ, то есть тестирование.

2. Основная часть

2.1. Аналитический обзор современных подходов к верификации распределенных систем

Настоящий обзор фиксирует современное состояние теории и методов верификации распределенных систем. Такая фиксация необходима для развертывания исследований в данном направлении. В конечном итоге результаты исследования позволят решать вопросы повышения надежности, отказоустойчивости, соответствия стандартам распределенных систем разных видов. В методологическом плане исследование нацелено на систематизацию различных аспектов функционирования распределенных систем, на асинхронность, надежность, защищенность. При этом будут исследованы методы спецификации отдельных аспектов и методы верификации, которые с одной стороны, позволяют фокусироваться на отдельных аспектах, и, с другой стороны, рассматривать их в совокупности.

2.2. Цели и рамки обзора

Задача разработки РС является одной из сложнейших задач, которые стоят в настоящее время перед информатикой и программной инженерией. Проблема верификации РС – это одна сторон разработки РС в целом, и состояние верификации РС также дает больше вопросов, чем ответов, предлагающих общие решения. Сложность проблемы обусловлена не только такими очевидными свойствами РС как большие размеры, географически удаленные компоненты, ненадежные каналы связи между компонентами, параллельные процессы, ассинхронные взаимодейтствия, сложные схемы синхронизации, поддержки глобального времени; сложность обусловлена еще и тем, что сейчас нет ясной и общепринятой систематизации характеристик функциональности РС. Это приводит к тому, что разные исследователи рассматривают разные аспекты функционирования РС в отрыве друг от друга, что в целом ведет к неадекватным моделям РС и неэффективными инструментам, которые используют эти модели. Простейший пример – это раздельное рассмотрение аспектов производительности и функциональной корректности. Это приводит к тому, что при тестировании производительности основное внимание уделяется тому, как система справляется с высокой нагрузкой (проверяется устойчивость) и не проверяется, насколько корректны результаты системы при работе в экстремальных режимах.

К сожалению нет не только готовых решений проблем разработки и верификации РС, нет и общих подходов к их решению. В связи с этим цель обзора собрать сведения о многообразии подходов к задачам верификации РС, с тем чтобы отталкиваясь от этой базы построить концептуальный базис, который позволит описывать требования к РС, их свойства, строить модели, анализировать свойства моделей и сопоставлять свойства модлеи и реализаций.

Среди подходов к анализу РС рассмотрены формальные модели, имитационное моделирование (simulating), измерения реальных систем.

Отправной точкой для любого метода верификации является выбор формализма (мета-модели), который будет использоваться для построения моделей. Рассмотрены следующие подходы: исполняемые модели, ограничительные модели (пропозициональные логики и исчисления предикатов, временные логики), аксиоматические модели (алгебраические и коалгебраические теории, исчисления параллельных процессов), а так же ряд иных подходов.

Из методов верификации, которые ограничиваются задачей установления корректности моделей РС, рассмотрены методы проверки модели (model checking), включая символьную (статическую) и динамическую проверку моделей, и вероятностную проверку моделей.

Проверка моделей дает ответ на вопрос, корректной ли является модель, на основе которой строиться РС. Для того, чтобы ответить на вопрос о корректности собственно РС, необходим статический и/или динамический анализ, то есть тестирование.

В рамки обзора вошли работы, которые дают важный вклад в исследование вопросов верификации РС. Хронологические рамки обзора – 1975-2003 годы.

2.3. Методы и техники анализа систем

2.3.1. Подходы к анализу распределенных систем

Существует большое количество подходов к анализу распределенных систем. Все их можно разделить на три основные категории – теоретические модели, имитация и непосредственные замеры показателей работающей системы.

Формальные модели

Преимущества были приведены ранее – система может быть проанализирована с математической точностью и обеспечить более правильные результаты с меньшими усилиями, чем при других видах анализа. Однако сложное динамическое поведение распределенных систем иногда трудно поддается моделированию, в частности, они могут быть гетерогенными.

Имитация

Имитация является средством анализа систем, хорошо зарекомендовавшим себя за продолжительное время. Шеннон приводит следующее формальное определение имитации:

Имитация – это процесс проектирования модели реальной системы и проведения экспериментов с этой моделью либо для понимания системы, либо для апробации различных стратегий (в границах, определяемых критерием или набором критериев) для работы системы.

Прямые замеры реальной системы

Эксперименты с физической системой имеют преимущества реализма. Нет никаких упрощающих предположений, которые могли бы свести на нет результаты. Однако, некоторые свойства реальной системы уменьшают это преимущества:

- Аппаратные замеры слишком специфичны, дороги и в них отсутствует гибкость.
- Этими замерами в систему может быть введена интерференция.
- Реальная система слишком сложна, чтобы контролировать каждый тик таймера и прерывание. Как следствие, эксперименты нельзя повторить, и даже ненастоящая нагрузка влечет к различным вариантам прогонов.

Особый интерес в рамках данного подхода представляют формальные методы подхода к верификации и тестированию, поскольку только то, что поддается алгоритмизации, можно, в принципе, автоматизировать с помощью компьютера.

Два дополняющих друг друга подхода к моделированию показали свое хорошее применение на практике – моделирование, основанное на состояниях, и моделирование, основанное на сценариях. Диаграммы состояний UML стали популярными в качестве техники описания поведения экземпляров классов в объектно-ориентированных системах. Формализмы, основанные на состояниях, также широко используются для моделирования распределенных и параллельных систем, в частности из-за того, что соответствующие модели могут быть строго проанализированы с помощью проверки модели. Практики также широко используют нотации и инструменты, основанные на сценариях. Варианты использования, диаграммы взаимодействия и последовательностей играют заметную роль в извлечении требований, основанных на сценариях. Стандарт ITUmsc (International Telecommunication Union message sequence chart) определяет нотацию сценария для детальной спецификации поведения телекоммуникационных систем. Хотя существует огромное количество исследований и по сценариям, и по конечным автоматам, отношение между ними еще нельзя считать полностью понятным, что еще более важно, полностью разработанным. Комплиментарная природа сценариев и конечных автоматов подсказывает несколько направлений для комбинации сильных сторон этих двух подходов к моделированию.

Формальная верификация включает модель системы, которая обычно представляет формально. Рассмотрим подходы к моделированию, существующие в сообществе разработчиков распределенных систем.

2.3.2. Классификация моделей

Детальное описание формализмов приводится в следующем разделе, здесь же мы ограничиваемся только их кратким описанием.

Исполнимые модели.

К ним можно отнести конечные автоматы (FSMs), расширенные конечные автоматы (EFSMs) и X-машины, взаимодействующие автоматы (CFSMs, CEFSMs), системы помеченных переходов и автоматы ввода-вывода (LTSs, IOSMs), временные автоматы, машины абстрактных состояний (ASMs), сети Петри, сценарные подходы (Use cases, MSC и пр.). Что касается ASMs, то для них пока не существует продвинутых предложений для представления распределенных систем. Исполнимые модели предоставляют возможность анализировать динамические свойства системы до ее реализации. При развитии соответствующих техник эти модели, в принципе, могут служить основой для генерации кода, и далее для генерации тестов.

Ограничительные модели

К ним можно отнести пропозициональные логики, временные логики.

Аксиоматические модели

К ним можно отнести алгебраические и коалгебраические теории, исчисления параллельных процессов.

Другие модели

Вероятностные модели, комбинаторные модели, модели на основе показателей.

2.3.3. Методы проверки модели

Несомненно, самым популярным и продвинутым методом проверки модели на сегодняшний день является **проверки модели (model-checking)**.

Метод проверки модели впервые был предложен в начале 80-ых как метод автоматической формальной верификации систем. Проверка модели является алгоритмическим методом для проверки, удовлетворяет ли система (обычно обогащенный автомат) некой спецификации (например, в виде формул темпоральной логики). В последнее время стал устоявшейся индустриальной практикой, и широко используется в практических приложениях, в особенности для проверки аппаратуры и коммуникационных протоколов, для анализа гибридных систем.

Проверка модели имеет две проблемы – проблему взрыва пространства состояний и много времени требуется для проверки даже простого логического свойства.

Метод заключается в специфицировании свойств системы в виде формул темпоральной логики или TCTL, построении модели в виде конечного автомата, автоматической проверке того, что эта модель удовлетворяет спецификации, причем при отрицательном выводе вырабатывается контрпример. Метод проверки модели состоит в переборе всех возможных переходов автомата из одного состояния в другое из некоего начального состояния системы. Все возможные трассы из начального набора состояний перебираются, чтобы убедиться в том, что все они являются безопасными или доказать, что живучесть системы не может быть нарушена.

Метод проверки модели страдает недостатком, широко известным под именем “взрыв количества состояний“.

При моделировании времени как непрерывной сущности даже самая простая модель имеет бесконечное число состояний. Наиболее известным подходом к решению этой проблемы является **метод символической проверки модели**, в котором проблема упрощается с помощью формирования классов эквивалентности. При этом используются компактные булевские формы для представления наборов состояний и переходов, как, например, BDDs (Binary Decision Diagrams) и накладываются некоторые ограничения на структуру пространства состояний.

Проверка модели является техникой, которая требует поддержки инструментами. Для временных автоматов существует ряд доступных зрелых инструментов. Каждый из них слегка отличается семантикой временных автоматов, однако это различие не касается существа выразительной мощности данного подхода. Синтез временных автоматов и проверки модели служит эффективной техникой верификации модели для параллельных

распределенных систем и в настоящее время является наиболее продвинутой техникой для анализа распределенных систем.

Существуют методы проверки, которые основаны на проверке модели или являются его расширением.

Вероятностная проверка модели является расширением техники проверки модели для вероятностных систем и впервые была предложен Хартом, Шерифом и Пнуэли .

2.3.4. Подходы к тестированию

Подходы к тестированию распределенных систем могут быть разбиты на два класса. Техники статического анализа и техники динамического анализа. Техники статического анализа можно разделить на три подгруппы: методы анализа параллельности, методы анализа потока данных и формальные методы. В рамках данного анализа основной интерес представляют методы тестирования на основе моделей (спецификаций).

Тестирование на основе моделей позволяет автоматизировать процесс генерации тестов из формальных спецификаций системы. Широко применяемые модели систем в тестировании используют конечные автоматы (FSM) и системы помеченных переходов (LTS). Эти модели служат для тестирования аппаратуры и тестирования протоколов на соответствие. Методы тестирование, основанные на таких моделях, в первую очередь фокусируются на генерации тестов, ориентированных на поток управления. Хотя эти методы хорошо зарекомендовали себя при тестировании аппаратуры и тестировании протоколов на соответствие, они не обладают достаточной мощностью для тестирования сложного поведения систем, зависящего от потока данных.

Связь между генерацией тестов и проверкой модели прослеживается в ряде исследований. Существуют подходы, расширяющие область применения проверки моделиа для формальной верификации систем, описанных с помощью конечных автоматов, до генерации тестов из такого описания, при этом проверка модели используется для генерации тестов, ориентированных как на потоки управления, так и на потоки данных.

Методы **аналитической верификации** являются более исчерпывающими, чем традиционное тестирование в том смысле, что они заменяют тестовые варианты символическим вычислением, которое покрывает целые области тестового пространства. Таким образом, аналитическая V&V часто вскрывает серьезные недостатки проектирования, которые выживают при экстенсивном традиционном тестировании. Примером может послужить ошибка в Pentium, касающаяся операций с плавающей точкой, которая осталась не выявленной на стадии тестирования первого выпуска Pentium. После такой дорогостоящей ошибки индустрия производства электронной аппаратуры вложила огромные суммы денег во внедрение аналитических подходов к отладке. Аналитические подходы к верификации, такие как проверка модели, показали свое успешное применение при проектировании и реализации микропроцессоров и параллельных протоколов.

Аналитическая верификация применима также к распределенным системам. Это автономное и бортовое ПО, которое оперирует в параллельном и распределенном окружении, ПО реального времени для авиации и т.д.

Тестирование в MDA (Model-Driven Architecture). MDA предложена OMG [MDA], и является стратегией, предназначенной для поддержки интероперабельности гетерогенных платформ промежуточных ПО (middleware). Стратегия предлагает переиспользование моделей независимых от платформы, основанное на различии и трансформации между платформо-независимыми и платформо-специфическими моделями. Применение этой стратегии называется модельно-управляемым подходом (Model-driven approach).

Хеккель и Лохман (R. Heckel и M. Lohmann) [HecLoh03] предлагают соответствующую **MDA** стратегию для тестирования и вводят новый термин model-driven testing. Эта стратегия тестирования требует похожей на **MDA** структуры для продвижения генерации тестовых вариантов и оракулов, а также выполнения тестов на различных целевых платформах. Метод исследуется на примере разработки и тестирования распределенных веб-приложений. Тестируются приложения, написанные при помощи model-driven approach. Model-driven testing берет свое начало от model-based testing. При этом генерируются платформо-независимые тестовые наборы и оракулы на основе платформо-независимой модели. После чего платформо-независимые тесты приводятся к целевой платформе (платформо-зависимые тестовые драйверы). В виде спецификаций выступают модифицированные UML-модели. Из use case, sequence diagrams и class diagrams строится UML-модель, отражающая статическую и динамическую модель системы. На основе спецификаций генерируется тестовый оракул. Для запуска тестов авторы используют модифицированный JUnit, на вход которому подается начальное состояние тестируемого метода и тестовые воздействия. Для организации тестовых драйверов используются Bridge design pattern для локального тестирования, и используются Bridge design pattern + Proxy design pattern для тестирования распределенного приложения.

В институте FOKUS (Fraunhofer) совместно с Ericsson, IBM, Motorola, Rational, Softeam и Telelogic проведен проект по созданию UML testing profile [TPTP]. Он основывается на UML 2.0 спецификациях (U2 patterns), позволяет писать спецификации статических и динамических аспектов поведения UML модели. Тесты могут преобразовываться в тесты на TTCN-3 и JUnit для переиспользования. Разработано расширение UML2 для описания различных артефактов тестовой системы. Создан язык для дизайна, визуализации, специфицирования, анализирования, построения и документирования артефактов тестовой системы. Сейчас Eclipse совместно с FOKUS реализует дополнение (add-on UML testing profile) для Eclipse.

2.3.5. Анализ производительности

В области анализа распределенных систем особое место занимает анализ производительности систем. В настоящее время научные исследования в

области анализа производительности нацелены на разработку формализмов и инструментов для моделирования систем и анализа показаний их производительности. Моделирование производительности может служить полезным источником экспертизы при анализе Марковских техник и численных вычислений, которые обычно не применяются в традиционной проверке модели. Кроме того, так как строится вероятностная модель, транзитные поведенческие аспекты, такие как вероятность доставки сообщений или качество сервиса, падающего ниже минимума внутри данного дедлайна, также могут быть проанализированы. Научные исследования в этой области представляют ряд техник, включающих измерение и тестирование, фокусирующихся на количественных характеристиках, и покрывают широкий спектр вопросов, например проектирование описательных языков (например, стохастические сети Петри и алгебры процессов), создание эффективных численных методов и инструментов для решения таких моделей и теории массового обслуживания.

2.3.6. Мониторинг

Различают мониторинг, основанный на событиях и основанный на времени.

Обычно подготовка системы к мониторингу означает внедрение датчиков (сенсоров) в соответствии с некоторой моделью мониторинга. Такая модель состоит из упрощенной модели системы и некоей спецификации, описывающей характеристики системы, которые должны наблюдаться.

Для описания процесса мониторинга было предложено большое количество моделей. Наиболее известными из них являются:

Модель управления событиями включает отчетность о текущем состоянии, регистрацию события и отчетность о нем, генерацию трассы (журнализацию). В модель не входят функции обработки трассы, такие как валидация трассы, комбинация и слияние трасс.

Парадигма событие/действие дает многоуровневую модель мониторинга

Общая функциональная модель мониторинга, которая основана на модели управления событиями с некоторыми изменениями и усовершенствованиями.

2.4. Формализмы и мета-модели, используемые для спецификации и анализа распределенных систем

2.4.1. Исполнимые модели

2.4.1.1. Конечные автоматы

Конечный автомат – это одна из основных моделей, применяемых при верификации программных систем. Конечные автоматы и их расширения активно используются для спецификации и верификации распределенных систем, особенно телекоммуникационных протоколов.

Конечный автомат содержит конечное число состояний, под внешними воздействиями совершает переходы между состояниями. При переходе конечный автомат может производить некоторое выходное воздействие.

Конечные автоматы широко используются для моделирования различных систем, включая электрические цепи, некоторые виды программных систем и коммуникационные протоколы.

Определение. Конечным автоматом называется пятерка M

$M = (I, O, S, \delta, \lambda)$ где

I, O, и S – конечные непустые множества входных символов, выходных символов и состояний соответственно.

δ: $S \times I \to S$ – функция перехода;

λ: $S \times I \to O$ – функция вывода.

Получив в состоянии $s \in S$ на вход символ $a \in I$, автомат переходит в состояние, задаваемое функцией переходов, $\delta(s, a)$, и выдает символ, определяемый функцией вывода, $\lambda(s, a)$.

2.4.2. Применения конечных автоматов

В настоящее время конечные автоматы и их расширения используются преимущественно для спецификации и верификации поведенческой части коммуникационных протоколов, так как для протоколов язык «состояние – переход» является естественным средством описания поведения.

Конечные автоматы используются для анализа возможных состояний программной системы и возможных эволюций состояний, выявления недостижимых состояний и блокировок.

2.4.3. Формализмы на конечных автоматах

2.4.3.1. Проверка моделей.

Аналитическая верификация

Тестирование. При тестировании конечные автоматы используются для построения оракулов и построения тестовых воздействий.

Тестирование соответствия. Важным применением конечных автоматов является тестирование реализаций протоколов на соответствие спецификации.

Ограничения применения конечных автоматов

Взрыв числа состояний. Основным препятствием на пути применения конечных автоматов к моделированию программных систем является так называемый «взрыв состояний». Как правило, состояние программной системы включает большое число переменных. Даже если у все переменные могут принимать лишь конечное число значений, общее число состояний может быть очень большим.

Один из подходов к решению проблемы взрыва состояний заключается в расширении семантики автоматной модели путем добавления в автомат новых элементов. Примерами являются расширенные конечные автоматы (EFSM), X-машины (X-machines),взаимодействующие конечные автоматы и

взаимодействующие расширенные конечные автоматы (CFSM и CEFSM), представленные в нашем обзоре.

Для построения автомата с меньшим числом состояний может применятся факторизация. При факторизации на множестве состояний исходного автомата задается отношение эквивалентности. По описанию исходного автомата строится автомат с построенными классами эквивалентности в качестве вершин.

Большое число входных воздействий. В практике часто встречаются системы с большим набором входных воздействий, что затрудняет анализ соответствующих конечных автоматов. Применяя факторизацию или расширяя автомат путем параметризации воздействий, можно построить автомат с меньшим алфавитом входных символов.

Недетерминизм. Многие реальные программные системы демонстрируют недетерминированное поведение, то есть при подаче одного и того же входного воздействия в одном и том же состоянии программная система может, вообще говоря, перейти в различные конечные состояния. Для моделирования таких систем были предложены различные расширения конечных автоматов, например недетерминированные конечные автоматы, вероятностные автоматы.

Ненаблюдаемые переходы. В программных системах переходы не всегда сопровождаются наблюдаемыми извне системы действиями. Для моделирования таких систем алфавиты входных и выходных символов автомата дополняют специальным пустым символом, который соответствует ненаблюдаемому воздействию, или строят эквивалентный автомат, как правило, недетерминированный, в котором нет ненаблюдаемых переходов.

2.4.3.2. Расширенные конечные автоматы

Расширенные конечные автоматы были введены для моделирования систем с большим или бесконечным числом состояний.

Определение. *Расширенный конечный автомат* (Extended Finite State Machine, EFSM) – это пятерка

$$M = (I, O, S, X, T)$$

где I, O, S, X, и T – это конечные множества входных символов, выходных символов, состояний, переменных и переходов соответственно. Каждый переход t представляет собой T упорядоченный набор:

$$t = (st , qt , at , ot , Pt , At)$$

где st , qt , at, и ot – это начальное (текущее) состояние, конечное (следующее) состояние, входное воздействие и выходное воздействие соответственно. Pt (x) задает на множестве переменных некоторый предикат, At (x) определяет действие над переменными.

В начале автомат находится в некотором исходном состоянии $s_1 \in S$, в котором переменные заданы значением: x_{init}.

Предположим, что в состоянии s набор переменных равен x. Под внешним воздействием a, автомат осуществляет переход $t = (s, q, a, o, P, A)$ если x удовлетворяет предикату P: $P(x) = \text{TRUE}$. В этом случае автомат выдает символ o, производит действие над переменными $x := A(x)$, и переходит в состояние q.

Если по заданным начальному состоянию, набору переменных и входному символу конечное состояние перехода определяется однозначно, то такой расширенный конечный автомат называется детерминированным конечным автоматом. В противном случае автомат называется недетерминированным.

Для систем с большим числом входных или выходных воздействий можно ввести параметризацию входных и выходных воздействий.

Расширенные автоматы широко используются для спецификации и тестирования узлов распределенных систем и поведенческой части коммуникационных протоколов.

Ограничения применения расширенных конечных автоматов.

Ограничения на размер системы. Расширенные конечные автоматы удобны для моделирования узлов распределенной системы, которые используют для взаимодействия протокол умеренной сложности (число состояний, переменных, мощности алфавитов входных и выходных символов измеряются десятками). При увеличении численных показателей, характеризующих модель, возрастает сложность аналитического исследования модели, затрудняется использование такой модели в тестировании.

Непосредственное применение расширенных конечных автоматов к моделированию сетей взаимодействующих узлов может привести к «взрыву числа состояний». Для расширенных конечных автоматов есть обобщения на сети взаимодействующих автоматов (CEFS), а так же различные подходы к уменьшению числа состояний – редукция, минимизация, факторизация.

Ненаблюдаемые переходы и недетерминизм. Существует ряд методов применения EFSM к тестированию недетерминированных программных систем.

2.4.3.3. Взаимодействующие конечные автоматы. Сети взаимодействующих конечных автоматов.

Для моделирования распределенных систем как сетей взаимодействующих узлов используют автоматные модели, которые получили название *взаимодействующие конечные автоматы* (Communicating Finite State Machine, CFSM) и *взаимодействующие расширенные конечные автоматы*[1] (Communicating Extended Finite State Machine, CEFSM).

Неформально взаимодействующий конечный автомат можно описать как конечный автомат, который считывает символы входного алфавита из

[1] В данном разделе конечные автоматы и расширенные конечные автоматы именуются просто конечными автоматами.

некоторых входных очередей и пишет символы выходного алфавита в некоторые выходные очереди.

Сети взаимодействующих конечных автоматов определяют как набор конечных автоматов, связанных очередями событий (каналами). Каждая очередь связывает ровно два автомата в сети, причем один из этих автоматов пишет в очередь, а второй читает, то есть очереди сообщений рассматриваются однонаправленными.

Формальные определения взаимодействующих конечных автоматов и сетей взаимодействующих конечных автоматов довольно громоздки, поэтому мы в данном обзоре их не приводим.

Взаимодействующие конечные автоматы используются для спецификации и верификации коммуникационных протоколов и распределенных систем. На основе модели взаимодействующих конечных автоматов были разработаны формализмы, которые получили широкое распространение – SDL и ESTELLE.

2.4.4. Ограничительные модели

Основаны на представлении свойств программы в виде ограничений (constraints) (предусловия, постусловия, инварианты типов данных, внутренние утверждения (assertions), инварианты циклов и пр.)

2.4.4.1. Пропозициональные логики и исчисления предикатов

Логикой Хоара называется формальное исчисление, рассматривающее высказывания вида

```
{<formula>}<program>{<formula>}
```

и дающее аксиомы и правила вывода для таких высказываний. Первоначальная версия логики Хоара основывалась на исчислении предикатов первого порядка.

Имеется множество подобных исчислений с разными аксиоматиками и различной выразительной силы, в том числе, как утверждается, и VDM (Vienna Development Method) формализуется в виде такого исчисления.

В этом же стиле сделаны языки Z и B (упрощенный Z), а также OCL (часть UML), SCR (Software Cost Reduction).

Иногда вводятся элементы исчислений более высоких порядков. Для параллелизма вводятся дополнительные правила, которые позволяют доказывать свойства программ, построенных в рамках CSP (см. далее, про исчисления параллельных процессов).

Пропозициональные логики используются для формального представления требований к целевой системе. На их основе проводится валидация этих требований, т.е. проверка их на полноту, непротиворечивость и т.п. В последнее время формальные спецификации, заданные в такой форме используются для генерации оракулов – программ, проверяющих корректность реакций системы на стимулы. Эти оракулы используются для тестирования.

2.4.4.2. Временные логики

Первоначально строились с целью введения в рассуждения понятия времени.

Используются для описания ограничений на возможные последовательности событий (LTL – Linear-time Temporal Logic, CTL – Computation Tree Logic, TLA – Temporal Logic of Actions, Unity, и пр.) и явных временных ограничений на события, абсолютных или относительных, (ITL – Interval Temporal Logic, Duration Calculus, и пр.).

Вообще, различных временных логик настолько много, что само перечисление всех упоминаемых – уже непростая задача.

Отцом-основателем их как логик считается Prior, основными продолжателями его дела в области Computer Science – Pnueli, Lamport и Burstall.

Временные логики используются как сами по себе, так и для усиления ограничений, задаваемых пропозициональной логикой, в частности, логикой Хоара. Применяются временные логики практически так же, как и пропозициональные. То есть для валидации требований и для генерации оракулов при тестировании.

Временные логики используются для моделирования, спецификации и верификации распределенных систем реального времени. Эти логики широко применяются благодаря тому, что они существенно облегчают описание временных свойств реальных систем.

Временные логики являются расширениями темпоральных логик, разработанных для специфицирования систем реального времени.

Использование темпоральных логик в качестве формализма для специфицирования поведения систем реального времени впервые было предложено Пнуэли. Темпоральные логики обеспечивают лаконичный и естественный способ выражения необходимых временных требований для систем, не зависящих от скорости, таких как инвариантность, предшествование и сохранение активного состояния. Традиционные темпоральные операторы, однако не могут измерять время и, следовательно, не могут качественно описать временные ограничения, необходимые для специфицирования систем реального времени, поэтому бурное развитие получили временные логики, включившие в свою семантику и синтаксис измерение времени.

Темпоральные логики являются классом модальных логик, в которых модальные операторы необходимости (☐) и возможности (✧) уточняются в темпоральной манере:

- базовый оператор ☐ интерпретируется как "всегда",
- базовый оператор ✧ означает "со временем" ("eventually").

Существующее многообразие темпоральных логик, слегка отличающихся синтаксисом и семантикой, которые применялись для специфицирования систем реального времени, описывается в [Eme90]. Любая из них может быть расширена конструкциями, которые описывают временные ограничения. Эти логики либо являются логиками первого порядка, либо имеют фрагменты пропозициональной логики. Логики, которые используются в сообществе

систем реального времени, могут быть разделены на логики линейного времени и логики временного ветвления. Это различие важно для выбора семантики трассы для системы реального времени.

Логики линейного времени интерпретируются через линейные структуры состояний. Каждая последовательность состояний представляет выполнение реактивной системы. Классическим примером такой логики является PTL (propositional linear-time temporal logic).

Логики временного ветвления интерпретируются через структуры состояний в виде деревьев. Каждое дерево представляет реактивную систему, чьи возможные последовательности выполнения соответствуют путям в дереве. от Branching-time temporal logics. Классическими примерами такой логики являются UB, CTL, и CTL* .

Логика линейного времени применяет наблюдательно-орентированную семантику, в то время как логики временного ветвления использует состояние-орентированную семантику.

Кроме этого, существуют еще различия в выборе темпоральных операторов. Темпоральным оператором, который используется наиболее часто, является оператор *until*. Также применяется оператор *next*. Некоторые включают оператор *since*, двойственный оператору *until*.

При расширении темпоральной логики до временной выделяются три подхода

Ограниченные темпоральные операторы.

Производится замещение неограниченных темпоральных операторов их версиями, ограничивающими время.

Фиксированная квантификация.

Введение квантификаторов, которые связывают переменные со временем.

Явные таймерные переменные.

При описании существующих временных логик рассматриваются следующие вопросы:

- является ли она логикой первого порядка или пропозициональной логикой,
- линейного времени или разветвленного времени,
- какие темпоральные операторы используются,
- в каком виде записываются временные ограничения,
- какие примитивные операторы применяются при записи временных ограничений.

Логики линейного времени

MTL (metric temporal logic) – пропозициональная логика с ограниченными операторами, включает версии темпоральных операторов *until*, *next*, *since* и *previous.*

TPTL (timed temporal logic) – пропозициональная логика полупорядка, которая использует только темпоральные операторы *until* и *next*.

RTTL (real-time temporal logic) – это логика первого порядка с явными таймерами.

XCTL (for explicit-clock temporal logic) – пропозициональная логика с явными таймерами, атомарные временные ограничения содержат сравнение и добавление.

MITL (metric interval temporal logic) оперирует неотрицательными вещественными числами в качестве домена времени, является пропозициональной логикой с основанной на интервалах строго монотонной семантике реального времени, т.е. интерпретируется через последовательности временных наблюдений.

Логики разветвленного времени.

RTCTL (for real-time computation tree logic) – пропозициональная логика разветвленного времени для синхронизированных систем, является расширением CTL с точечным представлением строго монотонного реального времени.

TCTL (timed computation tree logic) – пропозициональная логика с менее ограниченной семантикой, является расширением CTL с точечным представлением строго монотонного реального времени. Последующие версии этой логики используют интервальную семантику времени и синтаксис полупорядка с примитивами сравнения и добавления констант.

Для решения задач верификации распределенных систем на основе временных логик создаются языки, позволяющие описать временное поведение системы. При этом подходе возникает проблема разрешимости языка, которая зависит от временного домена и операций над временными переменными. Только пропозициональные версии темпоральных логик дают возможность справиться с проблемой разрешимости.

Все эти логики были предложены в конце 80-ых, начале 90-ых годов. Время показало, что в наиболее зрелых на сегодняшний день инструментах, таких как UPPAAL, HyTech, Kronos, TauTester, в качестве временной логики применяется TCTL.

2.4.5. Аксиоматические модели

2.4.5.1. Алгебры процессов

Алгебра процессов впервые появилась около двадцати лет назад как техника моделирования для функционального анализа параллельных систем. Алгебры процессов успешно применяются для описания и доказательства корректности распределенных систем. Другие подходы к описанию таких систем игнорируют ограничения на ресурсы и предполагают либо максимальный параллелизм (неограниченные ресурсы), либо чистый интерливинг (чередование), например, при единственном ресурсе.

Сильными сторонами этого подхода являются:

- **Композициональность**, т.е. возможность моделировать систему как взаимодействие ее подсистем.
- **Формальность** – все термины в языке имеют точные значения.
- **Абстрактность**, т.е. возможность построить модели из детализированных компонент, игнорируя, если нужно, их внутреннее поведение.

Благодаря различным расширениям в настоящее время эта техника широко применяется для анализа распределенных систем, в особенности для оценки производительности.

Алгебры процессов являются абстрактными языками, которые используются для спецификации и проектирования параллельных систем. В алгебрах процессов системы представляются в виде набора сущностей, называемых *агентами*, которые выполняют атомарные *действия*. Эти действия являются строительными блоками языка и используются для описания последовательного поведения агентов, которые могут выполняться параллельно, а также для описания синхронизации и взаимодействия между ними. Семантика моделей дается обычно в виде систем помеченных переходов.

Существует две основных подхода к алгебраическому моделированию процессов:

- CSP (Hoare's Communicating Sequential Processes) [Hoa85],
- CCS (Milner's Calculus of Communicating Systems) [Mil80].

Выделяется также ACP (Algebra of communicating processes) [BK85].

Эти алгебры основаны на предпосылках, что двумя самыми существенными понятиями в понимании сложных динамических систем являются параллельность и взаимодействие. Наиболее выдающийся аспект алгебр процессов заключается в том, что они поддерживают модульные аспекты спецификации и верификации систем. Это можно сделать благодаря алгебраическим законам, которые формируют композиционную систему доказательств и, таким образом, становится возможным верифицировать систему в целом с помощью рассуждений о ее частях.

Алгебры процессов без семантики времени широко используются для спецификации и верификации систем. Для того чтобы расширить их пригодность для описания систем реального времени, было разработано несколько алгебр процессов реального времени с добавлением понятия времени и включением в них временных операторов.

Можно отметить следующие разновидности алгебр процессов, которые активно исследуются и широко применяются для анализа распределенных систем:

- **Темпоральные**

  Temporal CCS расширяет CCS фиксированными задержками и асинхронным ожиданием. Действия понимаются как мгновенные, и

время ортогонально деятельности системы, что не позволяет, например, анализировать производительность.

- **Вероятностные**

  Probabalistic CCS – учитывается неопределенность поведения системы с помощью вероятностного подхода.

  - **Стохастические** (SPA) – формально определяется композициональность, добавляется информация о длительности действий, которая дает возможность анализировать производительность.
- **Временные**

  Представление времени – дискретное или плотное (dense) – является основным вопросом в научных исследованиях в области систем реального времени, в том числе и в подходах к временным алгебрам процессов. Время вводится одним из трех способов – в виде явных временных задержек, привязыванием информации о времени к операции взаимодействия, или введением оператора таймаут.

Дискретные временные алгебры процессов распадаются на две категории:

- те, которые допускают только одно действие на дискретную единицу времени, примерами могут служить SCCS , CCSR ,
- и те, которые оперируют квантами времени, чтобы представить его течение, и допускают несколько действий внутри кванта, примерами являются временные расширения LOTOS'a .

Алгебры, использующие модель плотного времени, можно разделить на две группы

- те, которые привязывают информацию о времени к действиям, например, расширение CCS ,
- те, которые используют явные временные задержки, расширение CSP, расширение LOTOS'a, PARTY

При моделировании алгебры процессов часто используются логики процессов для выражения свойств и запросов. Такие логики позволяют выразить утверждения о изменении состояния. Наиболее продвинутой является модальное μ-исчисление.

Существуют временные расширения логики Ноара, которые дают возможность написать спецификацию в виде пре- и постусловий с информацией о времени. Некоторые из них основаны на временном расширении CSP, однако предлагаемый язык трудно применять на практике.

2.4.5.2. Алгебраические и коалгебраические теории

В этом пункте имеются в виду специфические аксиоматические теории для описания свойств ПО (для каждой системы — своя теория).

Основой здесь можно считать теорию абстрактных типов данных (ADT). Сюда же относится большое множество формальных языков, предназначенных (или

просто поддерживающих) описание типов данных в аксиоматическом виде или функциональных языков, например, Larch, ML (много разновидностей), ADL (Algebraic Design Language), Charity, OBJ (и его потомки), CASL (Common Algebraic Specification Language).

Связь алгебраических спецификаций и тестирования — в основном, в методе ASTOOT и в работах D. Hamlet.

2.4.6. Дополнительные виды моделей

2.4.6.1. Модели на основе показателей

Это – особая разновидность тестирования, похожая на метод проверки совпадения семантики у программы, скомпилированной при помощи оптимизатора, и той же программы, скомпилированной без оптимизации.

Подход основан на определении системы показателей целевого ПО — функций, вычисляющих по ПО какие-то скалярные значения, и наличии правильного набора значений этих показателей.

2.4.6.2. Вероятностные модели

Сюда относятся подходы, моделирующие параметры использования ПО, возможные ошибки в нем, возможные траты пользователей в результате возникновения этих ошибок, внешние факторы, влияющие на стоимость сопровождения ПО, на основе вероятностных моделей разного рода.

Часто в качестве таких моделей встречаются марковские цепи и байесовские сети.

2.4.6.3. Комбинаторные модели

Сюда относятся различные подходы к построению тестов из имеющихся данных путем их комбинирования: как построить тест для функций с двумя, тремя, и более аргументами, которые можно перебирать независимо, как построить тестовую последовательность из кусков, представленных отдельными сценариями и пр.

2.5. Заключение обзора

В рамки обзора вошли работы, которые дают важный вклад в исследование вопросов верификации РС. Хронологические рамки обзора – 1975-2003 годы. Библиография содержит более 150 публикаций.

Изучение подходов в верификации РС показывает большое разнообразие формализмов и техник, которые используются для моделирования, спецификации, анализа моделей и собственно РС и, одновременно, отсутствие не только общепринятых подходов и методов спецификации и анализа, но даже общепринятой систематизации и классификации как видов и возможностей РС, так и методов их спецификации и анализа. В этом можно найти как отрицательные моменты – ситуация в области достаточно запутана; так и положительные – область быстро развивается, что делает вложения в исследования еще более обоснованными.

Общая, качественная оценка состояния и перспективности различных направлений исследований в области верификации РС представляется следующей.

Ни один из видов анализа моделей и собственно РС не играет доминирующую роль, используются как методы аналитического анализа, так динамического – тестирование.

Инструментов и методик, которые позволяют провести строгую аналитическую верификацию реальных РС пока нет. Аналитические подходы, включая аналитическую проверку моделей, применяются только для упрощенных моделей или для крайне простых реализаций. Достаточно общих подходов, предлагающих синтетические решения, которые совместно используют аналитические и динамические методы верификации пока нет.

Общей проблемой является задача комплексного рассмотрения различных характеристик РС, в первую очередь: функциональных, темпоральных аспектов и аспектов безопасности.

Среди мета-моделей, использующихся для моделирования, спецификации и верификации, доминирующее положение занимают исполнимые модели, например, SDL, LOTOS, исполнимые модели UML, ASM и др. Однако "доминирующее положение" – не означает, что этот подход к моделированию в перспективе заменит все остальные. Исполнимые модели хороши для там, где удается ограничиться небольшими, обозримыми моделями. Кроме того, исполнимые хорошо композируются. Проблемой является масштабируемость и сопоставление свойств поведения исполнимых моделей между собой и между моделями и реализацией. Эти две проблемы, в сущности, определяет границы применимости исполнимых моделей.

3. Аналитический обзор инструментов и методик верификации распределенных систем, используемых в индустриальной практике

3.1. Языки формальных спецификаций, используемые в индустрии, и инструменты их поддержки

Языки формальных спецификаций – FDTs (Formal Description Technique) – были разработаны для поддержки стандартизации OSI (Open Systems Interconnection), поскольку обычно стандарты пишутся на естественном языке, и в них отсутствует однозначность определения стандартов. Estelle, LOTOS и SDL являются официальными FDTs для использования в стандартизации.

3.1.1. LOTOS

LOTOS (Language Of Temporal Ordering Specification) – это формальная техника описания (FDT), основанная на темпоральном упорядочении наблюдаемого поведения. Разработана как международный стандарт. Используется для формального описания распределенных систем параллельной обработки информации. Применяется для формального описания

сервисных определений и спецификаций протоколов разных слоев иерархической архитектуры OSI, описанной в стандарте ISO 7498, и связанных с ним стандартов, и тестирования на соответствие реализаций протоколов OSI и/или функций OSI. Возможно его применение для других распределенных систем, таких как телефонные коммутируемые сети. LOTOS стандартизован как ISO/IEC 8807 [ISO/IEC 8807]. Первоначально был основан на формальном спецификационном языке CCS (Calculus of Communicating Systems) [Mil89]. Затем некоторые понятия и концепции были добавлены из CSP (Communicating Sequential Processes) [Hoa85]. Типизация данных была введена позже, для чего был адаптирован язык абстрактных типов данных ACT ONE. Основными достоинствами являются математическая строгость, выразительная мощность и масштабируемость. Позволяет описывать параллельность, недетерминизм, синхронные и асинхронные взаимодействия. Поддерживает несколько уровней абстракции и спецификационных стилей. Улучшенная версия стандарта называется E-LOTOS (Enhancements to LOTOS) [E-LOTOS]. Модель основана на событиях. Обычной операционной семантикой является система помеченных переходов.

Наиболее продвинутые инструменты на основе LOTOS:

3.1.1.1. TIPPtool

TIPPtool [Her00] – инструмент моделирования прототипов для создания и апробации TIPP (Timed Processes and Performance Evaluation) моделей параллельных и распределенных систем [TIPP]. Поддерживает LOTOS-ориентированный входной язык. Применяет анализ функциональности на основе анализа достижимости, обеспечивает анализ Марковских процессов, которые лежат в основе TIPP спецификаций.

Работает на SUN и HP под UNIX/X11 и доступен для некоммерческих организаций. Поддерживает интерфейсы к инструменту PEPP, к инструменту ALDEBARAN из CADP [CADP].

3.1.1.2. CADP

CADP (Caesar/Aldebaran Development Package) [Gar02] – набор инструментов для специфицирования и верификации систем, моделируемых с помощью конечных автоматов и описанных на языках алгебр процессов (Lotos). Предлагает ряд современных функциональностей процесса разработки ПО, таких как компиляция, быстрое прототипирование, интерактивная и управляемая симуляция, верификация на основе проверки эквивалентности, проверки прямым порядком и проверки модели на основе темпоральных логик и модального μ-исчисления (AFMC – Alternation Free Modal μ-Calculus), генерация тестов. Языки, модели и техники верификации, встроенные в CADP, имеют широкую область приложений, позволяя CADP применять к протоколам, распределенным системам, встроенному ПО, мобильной телефонии, асинхронному ПО, криптографии, системам безопасности. CADP используется как в индустриальных компаниях, так и в академических институтах в научных и преподавательских целях.

CADP дает возможность писать спецификации на трех формальных языках:

LOTOS (International Standard 8807) как язык для описания протоколов высокого уровня. Инструментальная панель имеет два компилятора – CAESAR и CAESAR.ADT, которые транслируют LOTOS-спецификации в C.

Системы помеченных переходов (LTS) для описания протоколов низкого уровня, т.е. конечные автоматы, переходы которых помечены именами действий.

Промежуточный уровень предлагается описывать с помощью сетей взаимодействующих автоматов, т.е. конечных автоматов, выполняющихся параллельно и синхронизирующихся средствами рандеву.

Для проверки эквивалентности поддерживаются:

- Эквивалентность по путям (Branching Equivalence) [stanford.edu1],
- Обозримая эквивалентность (Observational Equivalence) [stanford.edu2],
- Сильная бисимуляционная эквивалентность (Strong Bisimulation Equivalence) [stanford.edu],
- Слабая бисимуляционная эквивалентность (Weak Bisimulation Equivalence) [stanford.edu].

Поддерживается графический интерфейс пользователя, графическая симуляция и трассировка. Имеется возможность генерации контр примеров и их визуализации. Платформы – Windows и Unix и им подобные. Лицензия на бесплатное использование запрашивается через веб-сайт.

Инструментальная панель CADP содержит несколько взаимосвязанных компонент:

- ALDEBARAN – инструмент для верификации взаимодействующих систем, представленных системами помеченных переходов (LTS).
- CAESER – компилятор, который транслирует поведенческую часть LOTOS-спецификаций или в C-программу (которая может быть выполнена или симулирована), или в LTS (которая может быть верифицирована с помощью инструментов бисимуляции и /или с помощью средств темпоральной логики).
- OPEN/CAESAR – интегрированная среда для симуляции, выполнения, верификации (в частности, оперативной) и генерации тестов.
- BCG (Binary-Coded Graphs) – это и формат для представления эксплицитных LTSs, и совокупность библиотек и программ, обслуживающих этот формат.
- XTL (eXecutable Temporal Language) – язык, разработанный для реализации различных операторов темпоральной логики.
- EUCALYPTUS (European/Canadian LOTOS Protocol Tool Set) – интегрированная среда для языка LOTOS, графический интерфейс

пользователя, написанный на Tcl/Tk, который интегрирует CADP с другими инструментами.

- SMI (Symbolic Model Interface) – библиотека, обеспечивающая символическое представление систем конечных автоматов, в частности, коммуникационных протоколов. Система описывается как сеть взаимодействующих процессов, каждый из которых является расширенным конечным автоматом. Такое представление позволяет применять к системе проверку модели для различных темпоральных логик или символическую минимизацию для проверки эквивалентности.

3.1.1.3. TGV

TGV (Test Generation with Verification technology) [Fer97] – инструмент генерации тестовых наборов для тестирования на соответствие протоколов. Позволяет создавать тестовые наборы в формате TTCN (Tree and Tabular Combined Notation) из спецификаций на SDL и LOTOS. Интегрирован в CADP и IF.

3.1.1.4. CWB – NC

CWB – NC (The Concurrency Workbench of New Century) – основное предназначение – проверка модели и проверка эквивалентности систем, модели которых описаны с помощью алгебры процессов. Система поддерживает анализ достижимости, проверку модели в мю-исчислении. Поддерживает ряд языков проектирования, включая CCS, версию CCS с приоритетами, временной CCS (Timed CCS или TCCS) [Che90] и Basic Lotos. Поддерживается два интерфейса пользователя – текстовый, выполняющийся на различных платформах (Windows и Unix и им подобные), и графический.

Для проверки модели поддерживаются логики:

- AFMC (Alternation Free Modal mu-Calculus),
- CTL (Computation Tree Logic),
- GCTL* (Generalized CTL*).

Для проверки эквивалентности поддерживаются:

- May equivalence [Gla89],
- Must Equivalence [Gla89],
- Сильная бисимуляционная эквивалентность (Strong Bisimulation Equivalence) [stanford.edu],
- Слабая бисимуляционная эквивалентность (Weak Bisimulation Equivalence) [stanford.edu].

3.1.2. SDL

Продукты и инструменты, использующие SDL в качестве языка описания систем.

3.1.2.1. IF

IF [Boz01] – открытая интегрированная среда для промежуточного представления и валидации распределенных систем. Эта среда способна поддерживать различные техники валидации, от интерактивной симуляции до автоматической проверки свойств, а также имеет возможности для генерации исполнимого кода и тестовых вариантов. Естественно, все эти функциональности не могут быть встроены в один инструмент, поэтому эта среда снабжена возможностью интеграции различных инструментов. Ради эффективности среда поддерживает несколько уровней представлений программ. Среда **IF** оперирует тремя уровнями представления программ – спецификационный, промежуточный и уровень семантической модели LTS. Основным входным спецификационным формализмом является SDL, но предусматривается связь с другими языками, такими как UML, LOTOS и PROMELA. Промежуточный уровень соответствует IF-представлению, в котором система представляется как совокупность процессов, взаимодействующих либо асинхронно через набор буферов, либо синхронно через набор каналов. Процессы представляются с помощью временных автоматов с дедлайнами, расширенными дискретными переменными. Переходы защищены командами, состоящими из синхронных/асинхронных входов и выходов, присваиваний и установок таймеров. Буфера имеют различные политики для поддержания очередей (fifo, стек, корзина и т.д.), могут быть ограниченными или неограниченными, надежными или с потерями. Есть возможность модифицировать абстрактные деревья IF-представления. Так как все переменные, таймеры, буфера и коммуникационные структуры являются эксплицитными, применимы высокоуровневые преобразования, основанные на статическом анализе или абстракции программы. Кроме того, можно реализовать трансляторы из IF-представления в другие спецификационные формализмы. Уровень семантической модели служит для доступа к LTS, представляющей поведение IF-программы.

Компоненты IF вместе с некоторыми внешними инструментами, с которыми существует сильная связь.

Спецификационный уровень. ObjectGeode [Verilog] – коммерческий инструмент, разработанный Telelogic, поддерживающий SDL, MSC и OMT – используется для реализации транслятора из SDL в IF-представление.

Промежуточный уровень реализует несколько алгоритмов, основанных на статическом анализе, для преобразований IF-спецификаций. Для трансляции IF-спецификаций в PROMELA используется инструмент IF2PML, разработанный в Eindhoven TU.

Уровень семантической модели. Используется CADP [Gar02] – набор инструментов для верификации LOTOS-спецификаций. Два из его модел-чекеров связаны со средой IF – ALDEBARAN, основанный на бисимуляции, и EVALUATOR, основанный на альтернативно-свободном мю-исчислении. Для обоих инструментов диагностические последовательности вычисляются на уровне LTS и транслируются обратно в MSC для визуализации на

спецификационном уровне. Используются также такие инструменты, как KRONOS [Daw96], модел-чекер для символической верификации TCTL-формул на взаимодействующих временных автоматах, TGV – генератор тестовых последовательностей для тестирования на соответствие распределенных систем.

3.1.2.2. Telelogic ObjectGeode

Telelogic ObjectGeode [Verilog] – набор инструментов для анализа, проектирования, верификации и валидации распределенных приложений и приложений реального времени через симуляцию, генерацию кода и тестирование. Области применения – телекоммуникационные, авиационно-космические, оборонные, автомобилные, медицинские системы и системы управления процессом. Поддерживает UML, SDL и MSC. Осуществляется автоматическая генерация тестов из моделей на SDL и MSC.

3.1.2.3. Telelogic Tau SDL Suite

Telelogic Tau SDL Suite [tausdl] – инструмент для разработки ПО реального времени. Основан на SDL и MSC. Есть возможность описывать требования на UML, которые затем конвертируются в концепции SDL. Пакет позволяет автоматическую генерацию тестов из SDL-спецификаций.

3.1.3. Estelle

Estelle [estelle] – стандарт с 1989, применяется для спецификации коммуникационных стандартов, служит основой для анализа систем и для автоматической генерации реализаций непосредственно из спецификаций. Техника основана на расширенной модели состояния-переходы, т.е. модели недерминистического взаимодействующего автомата, расширенного с помощью добавления языка Pascal. Estelle можно рассматривать как набор расширений ISO стандартизованного Pascal (ISO International Standard 7185), уровень 0, который моделирует указанную систему как иерархическую структуру взаимодействующих автоматов, которые могут выполняться параллельно и взаимодействуют через обмен сообщениями и с помощью разделения некоторых переменных. Инструмент – EDT (Estelle Development Toolset) (National Institute of Telecommunications, France)

3.2. Методики и инструменты на базе формализмов описания систем

3.2.1. Алгебры процессов

Теория алгебр процессов зарекомендовала себя как мощная математическая модель описания параллельности, которая дает возможность для представления и последующего анализа распределенных систем и алгоритмов, протоколов и других систем как в области вычислительной техники, так и за ее пределами. Формализм алгебр процессов позволяет моделировать вероятные сбои ресурсов, потребление мощности ресурсов и применять приоритеты ресурсов. При этом становится возможной оценка альтернативного поведения

при различных планировщиках реального времени и ограничениях на ресурсы. Синтаксис алгебр процессов позволяет описывать иерархические системы, что дает возможность затем применять к ним композиционные рассуждения, осуществлять их верификацию и анализ. Семантика алгебр процессов обеспечивает непосредственную структурную интерпретацию описанной системы в систему помеченных переходов. Если ограничиваться случаем конечных автоматов (в основном, это статические сети автоматов), доказательства могут быть полностью автоматизированы и сведены к проверке модели, в которой основополагающая модель сопоставляется со своими модальными свойствами. Эти свойства могут представлены логическими формулами или, собственно, как спецификации автоматов.

Широко известным подходом к параллельным вычислениям является подход, основанный на процессах и действиях. Процесс P_0 может совершать некоторые действия $A \in \{a_1,\ldots,a_n\}$ или создавать новые (порожденные) процессы $P_1,\ldots, P_m$ в каждой фазе своего жизненного цикла. Такие системы описываются в терминах алгебр процессов или логик процессов. Эта модель параллельных вычислений обладает высокой степенью абстрактности по отношению к аппаратной конфигурации реальной системы и динамичностью по отношению к числу процессов.

Существует две основные алгебры процессов:

- **CSP** (Hoare's Communicating Sequential Processes) [Hoa85],
- **CCS** (Milner's Calculus of Communicating Systems) [Mil80].

Выделяется также **ACP** (Algebra of communicating processes) [BK85].

Далее описаны инструменты, использующие CSP.

3.2.1.1. PVS

PVS (Prototype Verification System) – интерактивная система доказательства теорем, основанная на логике высокого порядка с предикатными подтипами, зависимыми типами, рекурсивными типами данных, рекурсивными и индуктивными определениями и параметрическими теориями. Применяется в области систем реального времени (CPB) и гибридных систем. Модель описывается на CSP. Проверка модели осуществляется с помощью логики разветвленного времени, применяется также проверка эквивалентности и доказательство теорем. Обеспечивается графический пользовательский интерфейс. Распространяется по лицензии. Платформы – Unix и подобные.

3.2.1.2. FDR

FDR (Failures-Divergence Refinement) [Ros94] – инструмент для проверки уточнений процессов, определенных в CSP. Есть возможность проверить детерминизм конечного автомата, что, в основном, используется для проверки свойств надежности. Применяется проверка эквивалентности следующих типов:

- «разрешающая» эквивалентность (May equivalence),
- «обязывающая» эквивалентность (Must Equivalence),

- «наблюдаемая» эквивалентность (Observational Equivalence).

Поддерживается графический интерфейс пользователя. Поддерживается генерация контр примеров. Распространяется бесплатно для некоммерческого использования, для коммерческого использования требуется лицензия. Платформы –Unix и подобные.

3.2.1.3. LTSA

LTSA (Labelled Transition System Analyzer) [doc.ic.ac] – инструмент для верификации параллельных систем. Автоматически проверяет, что поведение параллельной системы удовлетворяет требуемым свойствам. Поддерживается анимация спецификации для исследования поведения системы. Система моделируется как совокупность взаимодействующих конечных автоматов. Свойства системы моделируются в нотации конечных автоматов. LTSA совершает композиционный анализ достижимости, чтобы выявить нарушения специфицированных свойств. Каждый компонент спецификации описывается как система помеченных переходов (LTS), которая содержит все достижимые состояния и все возможные переходы. Однако, явное описание LTS в терминах состояний, набора действий и отношений переходов возможно только для небольших систем, поэтому LTSA поддерживает нотацию алгебры процессов FSP (Finite State Processes – разновидность CSP алгебры процессов) для лаконичного описания поведения компоненты. Инструмент позволяет просматривать графическую интерпретацию LTS, соответствующую FSP-спецификации. Для проверки модели поддерживается логика LTL. Поддерживается графический интерфейс пользователя, графическая симуляция и трассировка. Поддерживается генерация контр примеров и их визуализация. Распространяется бесплатно. Инструмент может выполняться как апплет на основном компьютере. Платформы – Windows и Unix и им подобные. Язык разработки – Java.

3.2.1.4. FC2Tools

FC2Tools вместе с графическим редактором Autograph – пакет инструментов для спецификации синхронизированных сетей автоматов и их верификация через проверку эквивалентности (сильная и слабая бисимуляции, разветвленная), поиск тупиков, проверку свойств надежности. Основной характеристикой является композиционный (иерархический) подход к описанию системы и проверке свойств. Доступны эксплицитное и символическое (BDD) представления. Контрпримеры можно просматривать и проигрывать на графическом представлении. Распространяется бесплатно. Платформы –Unix и подобные.

Далее описаны инструменты, использующие CCS.

3.2.1.5. Edinburgh CWB

Edinburgh CWB (Edinburgh Concurrency Workbench) – автоматизированный инструмент, обеспечивающий управление и анализ параллельных систем. Поддерживает различные виды проверки эквивалентности и проверку модели с применением различных семантик представления процессов.

Языками моделирования являются:

- CCS,
- SCCS (Synchronous calculus of communicating systems) – синхронное исчисление взаимодействующих систем,
- TCCS (Timed CCS) [Che90].

Для проверки модели поддерживается μ-исчисление Парка.

Проверяются следующие эквивалентности:

- эквивалентность «ветвей» (Branching Equivalence),
- «разрешающая» эквивалентность (May equivalence),
- «обязывающая» эквивалентность (Must Equivalence),
- «наблюдаемая» эквивалентность (Observational Equivalence).
- Сильная бисимуляционная эквивалентность (Strong Bisimulation Equivalence),
- Сильная Марковская бисимуляционная эквивалентность (Strong Markovian Bisimulation Equivalence),
- Слабая бисимуляционная эквивалентность (Weak Bisimulation Equivalence).

Поддерживается генерация контрпримеров. Распространяется бесплатно при специальных условиях. Платформы –Unix и подобные. Язык разработки – стандартный ML.

3.2.2. Стохастические алгебры процессов

Стохастические алгебры процессов выросли из CSP и CCS.

3.2.2.1. PEPA

PEPA (Performance Evaluation Process Algebra) (Jane Hillston) [Hil96] – формальный язык для моделирования распределенных систем. Модели являются композицией компонент, которые совершают индивидуальные деятельности или кооперируются на разделяемых компонентах. С каждой деятельностью связывается оценка скорости, с которой она может совершаться. PEPA-модели содержат информацию о продолжительности деятельности и, через политику состязаний, их относительные вероятности. Из этих моделей можно генерировать Марковские цепи непрерывного времени (CTMC). Затем используется линейная алгебра для разрешения модели в терминах равновесного поведения. Это поведение представляется как вероятностное распределение через все возможные состояния модели. Такое распределение редко является конечной целью анализа производительности; обычно интерес представляют измерения производительности, которые должны быть выведены из этого распределения через *структуру поощрений*, определяемую над CTMC.

Язык PEPA добавлен в инструменты Möbius Modeling Framework (Performability Engineering Research Group), Motorola Center for High-Availability System Validation (University of Illinois at Urbana-Champaign). PEPA

также поддерживается вероятностным модел чекером PRISM (University of Birmingham, England). Поддержка PEPA встроена в CADP (VASY group, INRIA Rhône-Alpes).

Наиболее продвинутым инструментом на основе PEPA является **PEPA Workbench** [GH94] – инструмент для функциональная верификация и оценки производительности.

3.2.2.2. EMPA

EMPA (Extended Markovian Process Algebra) [Ber96] – расширенная алгебра Марковских процессов с порождающе-реактивной синхронизацией. Композиционная структура спецификаций описывается через алгебраические операторы. Экспоненциально распределенные продолжительности действий управляются политикой состязаний. Действия с нулевой продолжительностью управляются с помощью порождающей политики предварительного отбора. Действия с неспецифицированной продолжительностью с помощью реактивной политики предварительного отбора. Порождающе-реактивная синхронизация. Выбор с применением вероятности и приоритетов. Модели непрерывного и дискретного времени.

3.2.2.3. TwoTowers

TwoTowers [Ber98] – функциональная верификация и оценка производительности.

На этапе компиляции помимо обычной визуализации ошибок осуществляется визуализация графов состояний-переходов с полным представлением состояний. Интегрированная верификация – Марковская бисимуляционная проверка эквивалентности. Функциональная верификация – проверка модели в μ-calculus/CTL и GCTL, строгая /слабая бисимуляционная проверка эквивалентности.

3.2.2.4. ProVer

ProVer (Probabilistic Verifier) [You02] – реализует алгоритм верификации, независящий от модели, предложенный Younes & Simmons, для верификации свойств реального времени систем дискретных событий. Система специфицируется на языке описания стохастических процессов SPDL, свойства выражаются в стохастической логике CSL. SPDL позволяет специфицировать однородные по времени обобщенные полу-Марковские процессы (GSMPs). Находится в свободном распространении для некоммерческих целей. Платформы – Unix и подобные. Язык разработки – C++.

3.2.3. Сети Петри

Сети Петри – это формальный и графический язык для моделирования систем с параллельностью. Разрабатывается с начала 60-х годов, когда его впервые определил Карл Адам Петри. Впервые была сформулирована общая теория для дискретных параллельных систем. Язык является обобщением теории автоматов, в которой может быть выражена концепция параллельно выполняющихся событий.

3.2.3.1. Artifex

Artifex [Artis] – коммерческий инструмент предназначен для проектирования и симуляции коммуникационных сетей – протоколов и оборудования для коммутации. Поддерживает графический язык, на котором можно описывать динамическое поведение систем, интегрированную среду разработки для симуляции дискретных событий и разработки систем реального времени. Язык основан на нотации расширенных сетей Петри.

3.2.3.2. PEP

PEP (Programming Environment based on Petri nets) – интегрированная среда разработки и верификации систем реального времени для различных языков спецификации и программирования. Все возможности инструмента доступны из графического пользовательского интерфейса. Верификация выполняется на автоматически генерируемых моделях различных нотаций сетей Петри. Модель может быть описана с помощью B(PN)^2 (Basic Programming Notation for Petri Nets – язык программирования с эксплицитной параллельностью, семантика дается в терминах сетей Петри высокого уровня), PBC (Petri Box Calculus – композиционная алгебра сетей Петри), PFA (Parallel Finite Automata – сеть конечных автоматов, обогащенных программируемыми аннотациями на дугах, взаимодействие осуществляется через синхронизацию), Petri Nets [daimi] (формальный и графический язык для моделирования систем с параллельностью) или SDL (Specification and Description Language) [sdl-forum]. Проверка модели осуществляется с помощью логик CTL (Computation Tree Logic) и LTL (Linear Time Logic). Обеспечивается графический пользовательский интерфейс, графическая спецификация, графическая симуляция и трассировка, а также генерация контр примеров. Находится в свободном распространении. Платформы – Windows и Unix и им подобные. Языки разработки – Tcl/Tk, C, C++, Java.

3.2.3.3. INA

INA (Integrated Net Analyzer) – пакет инструментов, поддерживающий анализ сетей Петри типа Позиция/Переход и раскрашенных сетей Петри. Содержит текстовый редактор для сетей, поддержку симуляции, поддержку редукции для сетей Позиция/Переход, модел-чекер для логики CTL, поддержку анализа информации о структуре сети, об инвариантах позиций и переходов, анализа достижимости и возможности покрытия графов. Эта информация затем используется для верификации свойств ограниченности, живучести, возможности повторных использований и т.д. Поддерживается генерация контр примеров. Находится в свободном распространении. Платформы – Windows и Unix и им подобные.

3.2.3.4. PROD

PROD – инструмент для анализа достижимости высокоуровневых сетей Петри. Разработан группой исследователей и студентов Технологического университета в Хельсинки. Осуществляет верификацию LTL-формул оперативно, CTL-формулы верифицируются только после генерации

пространства состояний. Поддерживается генерация контр примеров. Находится в свободном распространении под лицензией GNU. Язык разработки – C. Платформы – Windows и Unix и им подобные.

3.2.3.5. The Kit

The Kit (The Model-Checking Kit) – совокупность программ, позволяющая моделировать системы конечных автоматов, используя ряд языков моделирования, и верифицировать их, используя ряд чекеров, включая дедлок-чекеры, чекеры достижимости, и модел-чекеры для темпоральных логик CTL и LTL. Чекеры взяты из различных существующих инструментов, таких как PROD, SMV, или PEP, которые применяют современные техники, чтобы избежать проблему взрыва состояний, такие как символическая проверка модели (BDDs) и методы частичного порядка. Языки моделирования включают низкоуровневый язык Petri Nets, т.к. некоторые инструменты экспортируют модели в этот язык, B(PN)^2 – структурированный язык параллельного программирования и PFA (Parallel Finite Automata) – язык описания взаимодействующих конечных автоматов. Независимо от выбранного пользователем языка моделирования, все чекеры могут применяться к одной и той же модели, и нет необходимости моделировать систему в различных языках моделирования. Поддерживается генерация контр примеров. Находится в свободном распространении. Язык разработки – C. Платформы –Unix и подобные.

3.2.3.6. Truth

Truth – инструмент для верификации распределенных систем. Компилятор спецификационного языка SLC позволяет добавлять и поддерживать новые спецификационные языки. Цель инструмента заключается в том, чтобы послужить прототипом инструмента верификации, в особенности для тестирования новых концепций в спецификации, моделировании параллельности и алгоритмов проверки модели. Языки моделирования включают низкоуровневый язык Petri Nets и CCS. Для проверки модели применяется мю-исчисление AFMC (Alternation Free Modal μ-Calculus). Поддерживается генерация и графическая визуализация контр примеров. Находится в свободном распространении для академического использования. Язык разработки – Haskell, Java. Платформы –Unix и подобные.

3.2.4. 2.2.4. Продукты, использующие темпоральные логики

3.2.4.1. Time Rover

Коммерческие продукты **Time Rover** [time-rover] оперируют с исполнимыми спецификациями и связанными с ними приложениями. Разработчики видят для своих продуктов две области применения – это верификация [DrS03] и оценивание требований во время выполнения. Для того, чтобы спецификации были исполнимыми, разработана технология, обладающая следующими свойствами:

- *Мощный спецификационный язык*. Инструменты поддерживают некоторые формы линейной временной темпоральной логики, которые включают ограничения реального времени, ограничения временных рядов, создание многочисленных копий (multi-instancing), и операторы счета (counting operators) (LTL/MTL/MTLS).
- *Графический интерфейс пользователя* для описания требований, симуляции, компоновки и администрирования.
- *Мониторинг реального времени*, который подразумевает отсутствие последующей обработки данных и трасс, что позволяет не сохранять историю трасс входящих событий.
- *Обоснованная семантика для конечных последовательностей*. Темпоральная логика обычно определяется и используется в контексте входных последовательностей бесконечной длины, что является одной из причин, по которой она не применяется в традиционном тестировании, где выполняемая последовательность всегда имеет конечную длину.
- *Генерация кода* – создается код из темпоральных формул, который затем может быть добавлен в процесс на встроенном целевом компьютере с целью мониторинга, даже если этот целевой компьютер не может взаимодействовать с удаленным монитором темпоральной логики. Сгенерированный код также полезен для приложений, которые используют формальные спецификации внутри конструкций, управляющих исключительными ситуациями.

Симуляция:

Язык воздействий. Инструменты Time Rover поддерживают язык воздействий, который позволяет описывать воздействия, которые ведут к успешному выполнению или сбою.

Сериализация. Инструменты имеют опцию совершать сериализацию контролируемой трассы. Обычно для мониторов темпоральной логики, которые не сохраняют историю трасс (онлайн мониторы), если удаленный монитор прекращает работу по какой-либо причине, он теряет способность восстанавливать контролируемое состояние, т.е. при рестарте оценивание темпорального правила будет начинаться со времени 0. Сериализация составного состояния мониторов дает возможность восстановления без потери способности продолжать мониторинг правил, начиная с их правил с их последнего составного состояния.

Продукты:

Temporal Rover – генератор кода для темпоральных логик (LTL/MTL/MTLS). Исполнимый код генерируется из спецификаций, Оформленных в виде комментариев. Создает код на C, C++, Java, VHDL, Verilog, способен взаимодействовать с Matlab и Ada.

DBRover [Drus03] – графическая версия Temporal Rover'a для удаленного мониторинга. Является монитором темпоральных правил, написанных на LTL или MTL. Темпоральные правила создаются графически с помощью IDE. Эти правила для требований затем отлаживаются с помощью графического симулятора, Temporal Rover генерирует код, компилирует его и создает удаленную исполнимую компоненту DBRover'a. DBRover осуществляет мониторинг сообщений, взаимодействующих с целевым приложением и верифицирует заданные пользователем правила. Для оценивания ограничений реального времени поддерживаются две формы замеров – на стороне сервера и на стороне клиента. DBRover может осуществлять мониторинг приложений для таких платформ и языков, как Windows, Linux, Unix, Java, VxWorks, Matlab, и Ada. DBRover используется для целей верификации, а также для проверки требований бизнеса и безопасности. Языки моделирования – Ada, C, C++, Java, VHDL, Verilog. В графическом интерфейсе можно писать графические спецификации, осуществлять симуляцию и трассировку. Платформы – Windows и Unix и им подобные.

ATG-Rover [Dru85] и **MC-Rover** [Dru03]. ATG-Rover автоматически генерирует тестовые последовательности из формальных спецификаций [Art03]. Этот инструмент в настоящее время не доступен для пользователей, а его основополагающая технология используется в MC-Rover'е, который находится в стадии разработки. MC-Rover является модел-чекером реального времени, способным выполнять валидацию свойств реального времени ПО.

3.2.4.2. KRONOS

KRONOS [Daw96] – основной целью является поддержка методов верификации для систем реального времени. Для описания модели используется нотация временных автоматов (Timed Automata). Поддерживаются методы верификации, основанные на проверке модели, требования к которой записываются на подмножестве TCTL (логика разветвленного времени), а также на проверке эквивалентности с помощью метода Time-abstracting bisimulation. Поддерживается генерация контр примеров. Находится в свободном распространении. Платформы – Windows и Unix и им подобные.

3.2.4.3. SGM

SGM (State-Graph Manipulators) – инструмент для верификации систем реального времени, встроенных систем и систем на чипах (System-on-Chips – SoC). В качестве языков моделирования используются параллельные конечные автоматы (PFA – Parallel Finite Automata) и временные автоматы (Timed Automata). Проверка модели осуществляется CTL модел-чекером. Имеется графический интерфейс. Поддерживается генерация контр примеров. Бесплатная лицензия для академического использования. Платформы – Windows и Unix и им подобные. Языки разработки – C, Qt / C++.

3.2.4.4. STeP

STeP (Stanford Temporal Prover) – инструмент для верификации реактивных систем, систем реального времени и встроенных систем. Комбинирует алгоритмические и дедуктивные методы верификации. Обеспечивает верификационные правила и верификационные диаграммы для того, чтобы сократить доказательство того факта, что систем удовлетворяет темпоральным свойствам. В некоторых случаях верификационные условия могут быть сгенерированы автоматически. Существует интерактивный инструмент доказательства теорем для тех условий, которые не могут быть доказаны автоматически. Инварианты можно сгенерировать автоматически, чтобы поддержать дедуктивные и дедуктивно-алгоритмические методы. В качестве языков моделирования используется SPL (Simple Programming Language – Pascal-подобный язык с поддержкой параллельности и недетерминистского выбора). Проверка модели осуществляется на основе линейной темпоральной логики LTL. Имеется графический интерфейс. Поддерживается генерация контр примеров. Распространяется бесплатно. Платформы –Unix и подобные.

3.2.4.5. UPPAAL

UPPAAL [Lar97] – интегрированная среда для разработки, валидации и верификации систем реального времени, моделируемых как сети временных автоматов, расширенных с помощью типов данных. В качестве метода верификации используется проверка модели, основанная на логике разветвленного времени (подмножество TCTL). Имеется графический интерфейс, графическая спецификация, графическая симуляция и трассировка. Поддерживается генерация контр примеров. Распространяется бесплатно для некоммерческих целей. Платформы – Windows и Unix и им подобные.

3.2.4.6. Mocha

Mocha [Alu98] – интегрированная среда для спецификации и верификации ПО. Основная цель – разработка новых алгоритмов и подходов к верификации. Доступен в двух версиях – cMocha (Version 1.0.1) и jMocha (Version 2.0). Система специфицируется на языке реактивных модулей (Reactive Modules), который обеспечивает формальную спецификацию гетерогенных систем и синхронными, асинхронными компонентами и компонентами реального времени. Требования специфицируются на альтернативной темпоральной логике (Alternating Temporal Logic), которая расширяет популярную логику CTL. Для проверки модели служит символический модел-чекер, основанный на BDD. Для проверки инвариантов инструмент поддерживает как символический, так и перечислимый поиск. Верификация реализации осуществляется путем сравнения трасс для спецификационного и реализационного модулей. Имеется графический интерфейс, графическая симуляция и трассировка. Поддерживается генерация контр примеров. Распространяется бесплатно. Платформы – Windows и Unix и им подобные.

3.3. Инструменты проверки модели (model-checkers)

Методы верификации с трудом пробивают себе дорогу в индустриальную сферу разработки ПО. Самыми продвинутыми методами верификации на сегодняшний день являются методы, основанные на проверке модели (model checking). Трудности, возникающие при применении метода проверки модели, мешают стать этому методу господствующим направлением в верификации ПО. Применение метода проверки модели требуют от пользователя не только умения построить формальную модель приложения, но также пользователь должен обладать специализированными знаниями как о приложении, так и о технике метода проверки модели. В статье [Hol02] Хольцман и Смит предлагают метод автоматического построения модели из исходного кода. Предлагаемый метод также значительно упрощает процесс приведения в соответствие модели с эволюционирующим приложением. Уровень доскональности тестирования ПО, который достигается при таком подходе, значительно выше, чем при традиционных подходах к тестированию. В качестве инструмента для проверки сгенерированной модели применяется SPIN. Наиболее продвинутыми инструмента для проверки моделей являются:

3.3.1. SPIN

SPIN [Hol97], [SPIN] – инструмент формальной верификации распределенных систем. Разрабатывался в Bell Labs в группе в Computing Sciences Research Center, начиная с 1980. С 1991 года находится в свободном распространении. Spin может быть использован в трех основных режимах: как симулятор, позволяющий осуществлять быстрое прототипирование со случайным, управляемым или интерактивным моделированием, как верификатор, способный строго доказать истинность требований корректности, указанных пользователем (используется теория редукции частичного порядка для оптимизации поиска), как система обоснованной аппроксимации, которая способна подтверждать правильность больших по объему протоколов с максимальным покрытием пространства состояний. Все ПО написано на стандартном ANSI C, и работает на всех версиях Unix, Linux, Plan9, Inferno, Solaris, и Windows.

Особенности:

Spin предназначен для верификации ПО, но не аппаратуры. Для спецификации системы использует язык высокого уровня, который называется PROMELA (PROcess MEta-LAnguage). Применяется для трассировки логических ошибок проектирования распределенных систем, таких как операционные системы, коммуникационные протоколы данных, системы коммутации, параллельные алгоритмы, железнодорожные протоколы обмена сигналами и т.д. Инструмент проверяет логическую согласованность спецификации. Составляет отчет о дедлоках, неспецифицированных откликах, неполноте флагов, условий состязания за ресурсы и недозволенных допущениях об относительных скоростях процессов.

Spin работает оперативно (on-the-fly), это означает, что он избегает необходимости строить глобальный граф состояний, или структуру Крипке, в качестве предпосылки для верификации каких-либо свойств системы.

Spin может быть использован в качестве полной системы проверки модели на основе линейной темпоральной логики (LTL), поддерживающей все требования корректности, выразимые в темпоральной логике линейного времени, а также как оперативный верификатор свойств надежности и живучести. Многие из последних свойств могут быть выражены и верифицированы без использования LTL. Свойства корректности могут быть специфицированы как инварианты системы или процесса (с помощью формальных утверждений), как требования линейной темпоральной логики (LTL), как формальный автомат Бучи, или более широко, как обобщенные омега-регулярные свойства.

Инструмент поддерживает динамическое наращивание и сокращение числа процессов.

Spin поддерживает буферизованную передачу сообщений, передачу сообщений методом рандеву и взаимодействие через разделяемую память. Смешанные системы, использующие и синхронное, и асинхронное взаимодействие, также поддерживаются. Идентификаторы канала сообщений для рандеву и буферизованных каналов могут передаваться в сообщениях от одного процесса к другому.

Инструмент поддерживает случайное, управляемое и интерактивное моделирование, а также исчерпывающий и частичный методы доказательства, основанные либо на глубинном поиске, либо на поиске по методу “сначала вширь”. Инструмент специально проектировался с возможностью легкого масштабирования и эффективного управления системами больших размеров.

Для оптимизации верификации инструмент использует метод редукции частичного порядка и (необязательно) для хранения технику таблиц типа BDD.

Для верификации проектирования строится формальная модель с использованием языка PROMELA. PROMELA –это недетерминистический язык, не строго привязанный к Дейкстровской нотации командного языка и заимствующий нотацию для операций ввода/вывода из языка CSP Хоара.

Теоретическое обоснование для Spin, касающееся конечных автоматов, можно найти в [Var86]. Алгоритм хеширования (для верификации систем больших размеров – необязательный в Spin) детально обсуждается в [Hol98]. Алгоритм редукции частичного порядка описывается в [Hol94]. Алгоритм вложенного глубинного поиска, используемый в Spin, можно найти в [Hol96].

3.3.2. HyTech

HyTech [Hen97] **–** символический модел-чекер для гибридных систем. Основной целью является поддержка методов верификации. Язык моделирования – Hybrid automata. Для моделирования гибридных систем вводится понятие гибридного автомата – конечного автомата с конечным

числом вещественных переменных, которые изменяются непрерывно, но не с постоянной скоростью, как в обычном временном автомате, а со скоростью, выражающейся через дифференциальные уравнения и дифференциальные неравенства. Поддерживаются логики CTL (Computation Tree Logic) и Monitor automata. Есть возможность осуществлять графическую спецификацию и генерацию контр примеров. Находится в свободном распространении. Платформы – Windows и Unix и им подобные.

3.3.3. Cadence SMV

Cadence SMV (Symbolic Model Verifier) [McM93] – модел чекер, разработанный в Cadence Berkeley Laboratories для Windows и Unix. SMV может быть использован для обучения общим принципам проверки модели и верификации на основе уточнений. SMV снабжен двумя языками моделирования – расширенным SMV и синхронным verilog. SMV допускает несколько форм спецификации, которые включают темпоральные логики CTL и LTL, конечные автоматы, встроенные утверждения и спецификации уточнений. Поддерживает графический интерфейс пользователя и генерацию контр примеров. Платформы – Windows и Unix и им подобные.

3.3.4. NuSMV

NuSMV (new symbolic model checker**) –** символический модел-чекер, который является расширением SMV. По сравнению с SMV обеспечивает такие возможности как взаимодействие конечных автоматов, анализ инвариантов, реализация методов декомпозиции, проверка модели на основе LTL. Поддерживает графический интерфейс пользователя, генерацию и визуализацию контр примеров. Распространяется бесплатно для академических целей. Платформы – Windows и Unix и им подобные.

3.3.5. PRISM

PRISM (Probabilistic Symbolic Model Checker) [HKN03] – вероятностный символический модел-чекер, разработанный в университете Бирмингема, для анализа вероятностных систем. Поддерживает три модели – Марковские цепи дискретного времени (DTMCs), Марковские цепи непрерывного времени (CTMCs) и Марковские процессы регултрования (MDPs). Для описания моделей разработан язык, основанный на описании состояний системы, Инструмент транслирует описание системы на этом языке в соответствующую модель, вычисляет набор достижимых состояний. Затем осуществляется анализ модели. Для проверки модели применяются логики CSL (Continuous Stochastic Logic) и PCTL (Probabilistic Computation Tree Logic). Поддерживается графический интерфейс пользователя. Распространяется бесплатно при определенных условиях. Платформы –Unix и подобные.

3.4. Инструменты визуализации и анимации распределенных систем

Распределенные алгоритмы являются сложными абстрактными объектами, поэтому визуализация играет важную роль в улучшении их восприятия и понимания. Это становится особенно актуальным в случае некоторых проблем представления конфигурации системы в виде графа, известных как детерминистически неразрешимых в распределенных вычислениях [Met00], так что привлечение подходов, основанных на рандомизации, для решения этих проблем становится неизбежным. Ясно, что теоретически предсказать поведение во время выполнения вычислений, основанных на случайности, довольно трудно, однако его можно наблюдать. Существуют принципиально непредсказуемые системы, которые называются вычислительно неприводимыми. С помощью инструментов для визуализации и анимации распределенных систем становится возможным манипулировать и экспериментировать с распределенными алгоритмами. Примерами систем для визуализации большого количества семейств распределенных алгоритмов являются нижеследующие.

3.4.1. Visidia

Visidia (Visualization and simulation of distributed algorithms) [Bau03] – набор инструментов, основанный на Java, графический интерфейс которого позволяет построить сеть и прототипировать распределенные алгоритмы. Подход использует локальные вычисления на графе в качестве базовой модели для анализа распределенных алгоритмов. Описание локальных вычислений основано на модели графа переименования меток (graph relabelling system), разработанной Метивьером [Met99].

Рассматривается сеть процессоров с произвольной топологией, которая представляется в виде связного неориентированного графа, где вершины являются процессорами, а ребра означают непосредственные коммуникационные связи. Метки, связанные с вершинами и ребрами, модифицируются локально, на подграфе фиксированного радиуса в соответствии с некоторыми правилами, зависящими только от этого подграфа (локальные вычисления). Переименование делается до тех пор, пока не останется ни одного возможного преобразования. Полученная конфигурация называется нормальной формой. Два последовательных шага переименования называются независимыми, если они применяются к непересекающимся подграфам. В этом случае они могут применяться в любом порядке или параллельно. Для получения автоматической реализации распределенных алгоритмов, описанных средствами локальных вычислений в асинхронной системе с асинхронной передачей сообщений, используются рандомизированные процедуры для осуществления рандеву или локальных выборов.

Модель содержит фиксированное число процессоров, но обладает топологическим представлением сети, обеспечивая основу для распределенных вычислений. Граф является гомогенным, и все узлы выполняют одну и ту же

программу. Распределенные процессоры симулируются через Java-потоки. Visidia может выполняться на нескольких компьютерах, объединенных в сеть, при этом потоки реализуются на разных компьютерах. Используется библиотека Java RMI для расширения распределения Visidia. Добавлена возможность оперировать отказами. С помощью графического интерфейса Visidia можно симулировать отказ процессора или соединения, и управлять таким отказом через предлагаемые примитивы.

3.4.2. ParaGraph

ParaGraph [Hea94] – графический инструмент для визуализации поведения и анализа производительности параллельных программ, которые используют интерфейс передачи сообщений (MPI – Message-Passing Interface). Визуальная анимация, основанная на информации трассы выполнения, собирается во время реального выполнения программы на параллельной компьютерной системе с передачей сообщений. Результирующие данные трассы проигрываются графически, чтобы обеспечить динамическое изображение поведения параллельной программы, а также графические сводки ее суммарной производительности.

3.4.3. PARADE, POLKA, XTANGO

Группа систем для визуализации и анимации параллельных и распределенных программ – **PARADE** [Sta95], **POLKA** [Sta93], **XTANGO** [Sta92]. Все эти системы разработаны под руководством Джона Стаско. PARADE (PARallel program Animation Development Environment) – интегрированная среда разработки, обеспечивающая визуализацию и анимацию параллельных и распределенных программ. Дает возможность управлять трассой событий и мониторингом. Анимационной компонентой PARADE является набор инструментов POLKA, который позволяет визуализировать программы, написанные на различных языках программирования для большого количества разных архитектур**.** Является post mortem системой, преимущество которой состоит в том, что все, относящееся к выполнению системы, сохраняется, и затем может быть в любое время воспроизведено. Недостатком является невозможность полностью проанализировать данные и поведение системы во время выполнения, что ведет к упущению возможных ошибок.

POLKA – общецелевая анимационная система. Была разработана для анимации параллельных программ и вычислений, но также может быть использована для анимации последовательных программ. POLKA имеет своим предшественником систему XTANGO, но является более мощной и более гибкой. Поддерживает цвет, реальное время, плавную анимацию. Реализована на C++.

3.4.4. ZADA

ZADA – совокупность интерактивных графических анимаций некоторых алгоритмов из области распределенных вычислений и коммуникационных протоколов. В качестве основы была взята система Zeus.

3.5. Коммерческие продукты верификации и тестирования распределенных систем

Коммерческие продукты для распределенных систем, в основном, обеспечивают некоторые аспекты процесса разработки ПО, такие как редактирование, генерация кода и тестирование. Однако, они обычно ограничиваются базовыми верификационными методами (исчерпываюшая симуляция, обнаружение дедлоков и т.п.). К тому же, как правило, они являются “закрытыми” в смысле интеграции с ними продуктов других фирм и инструментов научно-исследовательских проектов.

3.5.1. Rational Test® RealTime

Rational Test® RealTime – продукт разработан для автоматизации тестирования и анализа СРВ, встроенных и сетевых систем и предлагает следующие возможности:

Создание тестовых сценариев, драйверов, сопровождения, заглушек и проб прямо из исходного кода для анализа времени выполнения прямо из исходного кода,

Выполнение тестов на целевом компьютере осуществляется из пользовательской среды разработки,

Результаты прогонов тестов загружаются в сводные отчеты с детальной информацией о покрытии кода (9 уровней),

Осуществляется обнаружение утечек памяти, профилирование узких мест и трассировка функций времени выполнения,

Поддерживаются связи между кодом, тестами и UML-моделями,

Поддерживается регрессионное тестирование

В качестве целевых компьютеров могут служить компьютеры всех уровней сложности и всех общеизвестных платформ – от 8-bit микрочипов до 64-bit RTOS. Продукт Rational Test® RealTime интегрирован со многими продуктами Rational, в частности Rational PurifyPlus RealTime и Rational Rose® RealTime.

3.5.2. Tau/Tester™

Tau/Tester™ (Telelogic's systems) – составная часть продукта Tau Generation2 – предназначен для интеграционного и системного тестирования программных интерфейсов (таких как TCP/IP) и аппаратных интерфейсов (таких как RS232). Применяется язык тестов TTCN-3 (Testing and Testing Control Notation v. 3). Для представления тестовых трасс используется редактор UML (2.0) диаграмм последовательностей. Тестирование интерфейсов компонент производится с использованием основанных на процедурах взаимодействиях (RPC, CORBA). Системные интерфейсы могут быть как с синхронными, так и с асинхронными взаимодействиями, такие как сетевые протоколы SIP (Session Initiation Protocol) и IPv6. Поддерживается распределенное тестирование. Возможна интеграция с системами конфигурационного управления. Осуществляется

автоматическая генерация тестовых вариантов. Генератор TTCN в XML дает возможность создавать тестовую документацию в XML формате.

3.5.3. Продукты фирмы I-Logix

Statemate [Klo01] – инструмент графического моделирования и симуляции для разработки сложных встроенных систем. Применяются некоторые UML диаграммы. Позволяет создавать исполнимую спецификацию за счет формальной связи между требованиями и реализацией. Создает визуальную графическую спецификацию, которая представляет требуемые функции и поведение системы. Спецификация может быть выполнена, или графически симулирована, позволяя выявить сценарии, в которых поведение и взаимодействие между элементами системы корректны. Эти сценарии сохраняются и включаются в тестовые планы, которые позже выполняются на целевой системе. Имеется генератор кода, который автоматически производит код приложения из графической спецификации, настроенный, в основном, на приложения автомобильной индустрии. Применяется формальная верификация, чтобы гарантировать, что графическая спецификация и соответствующий ей проект удовлетворяют свойствам, указанным пользователем, например, таким, как надежность. Statemate ATG (Automatic Test Generator) способен автоматически создавать тесты из графической спецификации (90-100% покрытие).

Rhapsody – платформа для визуальной разработки приложений, основанная на UML. Архитекторы ПО могут выполнять и обосновывать UML модели. Разработчики ПО могут осуществлять разработку приложений в графическом окружении, генерировать код из моделей в C, C++, Java и Ada. Rhapsody содержит полный набор UML-диаграмм, позволяет создавать документацию. Имеется возможность отладки на уровне дизайна на основной и целевой платформах. Поддерживается ассоциативная связь модель/код и интегрированная среда для развертывания встроенных приложений. Тесты могут быть автоматически сгенерированы из моделей.

3.5.4. Продукты фирмы MathWorks

Stateflow [Stateflow] – интерактивный инструмент проектирования для моделирования, симулирования и анализа систем, управляемых событиями. Интегрирован с Simulink и MATLAB (основа для всех продуктов фирмы MathWorks, которая является и языком, и интегрированной средой). В основе Stateflow лежит комбинация традиционных диаграмм состояния-переходы и диаграмм потоков управления. Схемы Stateflow дают возможность графического представления иерархических и параллельных состояний и переходов между ними, управляемых событиями. Stateflow дополняет традиционные схемы состояний потоками управления, графическими функциями, темпоральными операторами, рассылкой, управляемой событиями и поддержкой моделирования унаследованного C-кода. Используя Stateflow, можно разрабатывать визуальные модели систем, управляемых событиями, которые включают диаграммы состояний-переходов без знания теории

конечных автоматов. Можно генерировать С-код из моделей. Имеется возможность анимировать схемы для улучшения понимания системы и облегчения ее отладки. Обеспечивается проверка во время выполнения конфликтов переходов, проблем зацикливания, согласованности состояний, нарушений, связанных с диапазонами данных, и условий переполнения. Включает интегрированный отладчик для установки графических контрольных точек, хождения по моделям, просмотра данных, анализа покрытия диаграмм. Применяется для разработки встроенных систем в автомобильных, аэрокосмических и телекоммуникационных приложениях проектирования.

Simulink [Simulink] **–** графическая среда для разработки систем непрерывного и дискретного времени. Служит средой моделирования и прототипирования для моделирования, симуляции и анализа динамических систем. Обеспечивает интерфейс для описания блок-схем, построенный на численной, графической и программной функциональности ядра среды MATLAB.

3.5.5. Продукты фирмы Compuware

Reconcile [Compu] – система управления требованиями. Позволяет создавать, изменять, отслеживать требования к проекту и составлять о них отчеты.

***QA*Director** – интегрированная среда для автоматизированного тестирования распределенных крупномасштабных приложений на протяжении их полного жизненного цикла. При использовании вместе с Reconcile (система управления требованиями, позволяющая создавать, изменять, отслеживать требования к проекту и составлять о них отчеты) дает возможность генерировать тест-планы в соответствии с изменяющимися требованиями.

***QA*Run –** инструмент для автоматизированного функционального тестирования распределенных систем. Для автоматизации генерации тестовых сценариев используется объектно-ориентированный подход. Служит для тестирования веб-приложений, Java-приложений, а также клент-серверных и основанных на эмуляторах приложений. Поддерживается репозиторий для хранения сценариев, проверок, событий и определений объектов и ссылок между ними.

TestPartner – инструмент для автоматизированного функционального тестирования, который был специально спроектирован для тестирования сложных приложений, основанных на Microsoft, Java и web-технологиях. Позволяет создавать многократно используемые тесты с помощью визуализации сценариев и автоматических мастеров.

***QA*Center Performance Edition –** интегрированная среда тестирования производительности и мониторинга сервера для веб-приложений и распределенных клиент-серверных приложений. Позволяет записывать и проигрывать виртуальные пользовательские транзакции для воспроизведения точных промышленных симуляций и измерения производительности и масштабируемости приложений.

***QA*Load –** инструмент тестирования производительности приложений при больших нагрузках.

3.5.6. AGEDIS

AGEDIS Consortium [AGEDIS] (**A**utomated **G**eneration and **E**xecution of Test Suites for **DI**stributed Component-based **S**oftware) – коммерческая фирма, позиционирующаяся в области тестирования распределенных систем.

Методология тестирования **AGEDIS** включает

- создание модели поведения тестируемой системы;
- аннотацию модели информацией о тестировании – описание критериев покрытия, специальных тестовых назначений и ограничений тестирования;
- автоматическую генерацию тестового набора;
- проведение анализа модели, тестовой информации, тестового набора совместно с разработчиками и заказчиками;
- автоматическое выполнение тестового набора и журнализации результатов;
- анализ результатов и повтор шагов 2) – 5) до тех пор, пока не будут достигнуты цели покрытия и качества.

AGEDIS обеспечивает следующие инструменты для реализации своей методологии тестирования:

- Model Compiler – на вход принимает модель, описанную на AGEDIS Modeling Language (AML), и производит результат в стандартном промежуточном формате, который может воспринимать генератор тестового набора. Язык моделирования включает как модель поведения, так и директивы генерации тестов.
- Test Suite Generator – читает промежуточный формат и производит тестовый набор в формате Abstract Test Suite (ATS).
- Test Execution Engine – читает абстрактный тестовый набор и директивы выполнения тестов, выполняет тестовый набор на тестируемом приложении и журнализует результаты прогона.
- AML Profile – дает возможность инструменту моделирования Objecteering UML производить модели в AML.
- Браузер абстрактного тестового набора.

Разрабатываются инструменты – интегрированная платформа для всех инструментов **AGEDIS**, браузер отчетов, автоматизированные компоненты для анализа и повтора шагов методологии.

3.5.7. Conformiq Test Generator

Conformiq Test Generator™ [Conformiq] – коммерческий инструмент для автоматизированного тестирования на основе моделей (model-based testing) распределенныз систем, таких как системы контроля реального времени, телекоммуникационных систем и веб-приложений. Модели представлены UML диаграммами состояний с поддержкой свойств реального времени. Для

описания действий и структур данных вводится язык Action Language. Редактор UML моделей позволяет каждой модели содержать несколько конечных автоматов, которые в свою очередь состоят из нескольких уровней иерархии. Экземпляры конечных автоматов могут создаваться динамически, наподобие создания новых объектов в объектно-ориентированных языках. Работает в пакетном и динамическом режимах тестирования. Тестовые наборы генерируются автоматически в языки TTCN-3 и Java. В информации на сайте не указывается, какие директивы покрытия применяются, но упоминается, что инструмент включает измерения покрытия, достигнутого тестом.

3.5.8. AsmL Test Tool

AsmL Test Tool – интегрированная среда для тестирования на основе моделей **AsmL** [BGNSTV], которая пока еще находится в стадии прототипирования. AsmL (Abstract State Machine Language) [AsmL] – язык исполнимых спецификаций, основанный на теории Abstract State Machines [ASM]. Текущая версия, AsmL 2 (AsmL for Microsoft .NET), встроена в Microsoft Word и Microsoft Visual Studio.NET и использует XML и Word для написания спецификаций. Инструмент для генерации тестов, который работает с этим спецификационным языком, поддерживает как генерацию тестовых последовательностей, так и генерацию параметров. Модель ASM (потенциально бесконечная) сводится к конечному автомату (FSM), что и позволяет генерировать тестовые последовательности из FSM. Инструмент позволяет выполнять AsmL модель параллельно с конкретной реализацией для проверки на соответствие данной реализацией со спецификацией, этот процесс авторы называют верификацией времени исполнения для тестирования на соответствие. Алгоритм генерации тестов выполняет полное покрытие переходов полученного конечного автомата, при этом тестовые оракулы вырабатываются на основе модели. Эта интегрированная среда реализует полу-автоматический подход, требуя от пользователя в графическом редакторе аннотировать модели информацией для генерации параметров и последовательностей вызовов, а также формировать связывания между моделью и реализацией для тестирования на соответствие. Недетерминизм модели обрабатывается с помощью поддержки набора допустимых поведений модели, и эта проблема в данный момент решена только на уровне верификации во время выполнения.

3.6. Продукты мониторинга распределенных систем

Разработчики и пользователи распределенных систем часто сталкиваются с проблемами производительности, таких как непредвиденная низкая пропускная способность или высокая латентность. Определение источника таких проблем требует детального сквозного инструментария всех компонент системы, включая приложения, операционные системы, хосты и сетевое оборудование.

3.6.1. NetLogger

NetLogger (Networked Application Logger) **–** набор инструментов для настройки и отладки производительности распределенных систем, для обнаружения узких мест в таких системах [Gun03]. Разработан в Lawrence Berkeley National Lab. NetLogger – это и методология для детального анализа распределенных приложений, которая дает возможность диагностировать в реальном времени проблемы производительности сложных распределенных систем. Набор инструментов NetLogger включает инструменты для генерации журналов событий с временными метками для мониторинга на уровне приложений и на уровне системы, имеет средства визуализации состояния системы на основе данных из этих журналов. Подход NetLogger'а отличается тем, что он комбинирует мониторинг сетей, хостов и приложений, обеспечивая полное представление системы в целом. NetLogger наиболее часто использовался в слабосвязанных архитектурах клиент-сервер.

NetLogger Toolkit состоит из четырех компонент: API и библиотека функций, позволяющая упростить генерацию журналов событий уровня приложений, набор инструментов для накопления и сортировки лог-файлов, набор инструментов для мониторинга хостов и сетей, и инструмент для визуализации и анализа лог-файлов. Для создания журналов событий разработчики вставляют в свой код обращения к NetLogger API во всех критических точках кода и затем линкуют приложение с библиотекой NetLogger. Эта возможность доступна в нескольких языках currently: Java, C, C++, Python, и Perl. Все инструменты NetLogger Toolkit разделяют общий формат лог-файлов и предполагают существование точных и синхронизированных системных часов.

3.6.2. ARM

Инструментальный пакет – ARM (Application Response Measurement) API, Open Group Enterprise Management Forum [OGEMF02], – определяет функциональные вызовы, которые могут быть использованы в приложении для мониторинга транзакций.

3.6.3. Verisoft

Verisoft [God97], [God98] – анализ C-программ с помощью мониторинга непосредственного выполнения кода. Проверки ограничиваются базовыми свойствами надежности и не допускают верификацию общих темпоральных свойств. Обеспечивается графический пользовательский интерфейс, графическая симуляция и трассировка, а также генерация и визуализация контр примеров. Находится в свободном распространении. Платформы – Unix и подобные.

3.7. Таблица инструментов для верификации распределенных систем

	Purpose			Specific Features			Graphical Interface		
Name	Model Checking	Equiv. Checking	Theorem Proving	Real Time	Probabilistic	Hybrid	GUI	Graph. Specif.	Graph. Sim.
Cadence SMV	☑		☑				☑		
CADP	☑	☑			☑		☑		☑
CWB - NC	☑	☑		☑			☑		
DBRover	☑			☑			☑	☑	☑
Edinburgh CWB	☑	☑							
FC2Tools	☑	☑						☑	☑
FDR		☑					☑		
HyTech	☑			☑		☑		☑	
INA	☑								
KRONOS	☑	☑		☑					
LTSA	☑						☑		☑
Mocha	☑	☑					☑		☑
NuSMV	☑						☑		
PEP	☑			☑			☑	☑	☑
PRISM	☑				☑		☑		
PROD	☑								
ProVer	☑			☑	☑				
PVS	☑	☑	☑	☑		☑	☑		
SGM	☑			☑			☑		
SPIN	☑						☑		
STeP	☑		☑	☑		☑	☑		
Temporal Rover	☑			☑					
The Kit	☑								
Truth	☑								
TwoTowers	☑	☑			☑		☑		
UPPAAL	☑			☑			☑	☑	☑
VeriSoft	☑						☑		☑

	Purpose			Availability			Platforms		
Name	Model Checking	Equiv. Checking	Theorem Proving	Free	Free Under Cond.	Commercial	Win.	Unix & related	Others
Cadence SMV	☑		☑		☑		☑	☑	
CADP	☑	☑			☑		☑	☑	
CWB - NC	☑	☑			☑		☑	☑	
DBRover	☑					☑	☑	☑	
Edinburgh CWB	☑	☑			☑		☑	☑	☑
FC2Tools	☑	☑		☑				☑	
FDR		☑			☑	☑		☑	
HyTech	☑			☑			☑	☑	
INA	☑			☑			☑	☑	
KRONOS	☑	☑		☑			☑	☑	
LTSA	☑			☑			☑	☑	
Mocha	☑	☑		☑			☑	☑	
NuSMV	☑			☑			☑	☑	
PEP	☑			☑			☑	☑	
PRISM	☑				☑			☑	
PROD	☑			☑			☑	☑	
ProVer	☑				☑			☑	
PVS	☑	☑	☑		☑			☑	
SGM	☑				☑		☑	☑	
SPIN	☑				☑		☑	☑	
STeP	☑		☑	☑				☑	
Temporal Rover	☑					☑			☑
The Kit	☑			☑				☑	
Truth	☑			☑				☑	
TwoTowers	☑	☑			☑			☑	
UPPAAL	☑				☑		☑	☑	
VeriSoft	☑				☑			☑	

3.8. Результаты обзора

Результатом является аналитический обзор современных методик и инструментальных средств верификации распределенных систем, используемых в индустриальной практике. Были рассмотрены более 30 инструментов как прототипных, разработанных в академических кругах, так и промышленных. Этот обзор фиксирует современное состояние методов верификации распределенных систем и мощности их инструментальной поддержки. Такая фиксация необходима для развертывания исследований в данном направлении. Целью проекта в целом является проведение фундаментальных исследований в области создания распределенных систем (РС). В конечном итоге результаты исследования позволят решать вопросы повышения надежности, отказоустойчивости, соответствия стандартам распределенных систем разных видов. В методологическом плане исследование нацелено на систематизацию различных аспектов функционирования распределенных систем, на асинхронность, надежность, защищенность.

Задача разработки РС является одной из сложнейших задач, которые стоят в настоящее время перед информатикой и программной инженерией. Проблема верификации РС – это одна сторон разработки РС в целом, и состояние верификации РС также дает больше вопросов, чем ответов, предлагающих общие решения. Сложность проблемы обусловлена не только такими очевидными свойствами РС как большие размеры, географически удаленные компоненты, ненадежные каналы связи между компонентами, параллельные процессы, ассинхронные взаимодейтствия, сложные схемы синхронизации, поддержки глобального времени; сложность обусловлена еще и тем, что сейчас нет ясной и общепринятой систематизации характеристик функциональности РС. Это приводит к тому, что разные исследователи рассматривают разные аспекты функционирования РС в отрыве друг от друга, что в целом ведет к неадекватным моделям РС и неэффективными инструментам, которые используют эти модели. Простейший пример – это раздельное рассмотрение аспектов производительности и функциональной корректности. Это приводит к тому, что при тестировании производительности основное внимание уделяется тому, как система справляется с высокой нагрузкой (проверяется устойчивость) и не проверяется, насколько корректны результаты системы при работе в экстремальных режимах.

К сожалению нет не только готовых решений проблем разработки и верификации РС, нет и общих подходов к их решению. В связи с этим цель обзора состояла в том, чтобы собрать сведения о многообразии методов верификации РС и их инструментальной поддержки, с тем чтобы отталкиваясь от этой базы построить концептуальный базис, который позволит описывать требования к РС, их свойства, строить модели, анализировать свойства моделей и сопоставлять свойства моделей и реализаций.

Изучение методик и инструментов верификации РС показывает большое разнообразие формализмов и техник, которые используются для

моделирования, спецификации, анализа моделей и собственно РС и, одновременно, отсутствие не только общепринятых подходов и методов спецификации и анализа, но даже общепринятой систематизации и классификации как видов и возможностей РС, так и методов их спецификации и анализа. В этом можно найти как отрицательные моменты – ситуация в области достаточно запутана; так и положительные – область быстро развивается, что делает вложения в исследования еще более обоснованными.

Общая, качественная оценка состояния и перспективности различных направлений исследований в области верификации РС представляется следующей.

Ни одна из методик и, соответственно, ни один из инструментов верификации РС не играет доминирующую роль, используются как методики аналитического анализа, так динамического – тестирование.

Инструментов и методик, которые позволяют провести строгую аналитическую верификацию реальных РС пока нет. Аналитические подходы, включая аналитическую проверку моделей, применяются только для упрощенных моделей или для крайне простых реализаций. Достаточно общих подходов, предлагающих синтетические решения, которые совместно используют аналитические и динамические методы верификации пока нет.

Общей проблемой является задача комплексного рассмотрения различных характеристик РС, в первую очередь: функциональных, темпоральных аспектов и аспектов безопасности.

Проблемой остаются масштабируемость и сопоставление свойств поведения исполнимых моделей между собой и между моделями и реализацией. Эти две проблемы, в сущности, определяет границы применимости исполнимых методик.

4. Заключение

Общая, качественная оценка состояния и перспективности различных направлений исследований в области верификации РС, основанная на анализе состояния исследований и разработок в этой области, представляется следующей:

Ни одна из методик и, соответсвенно, ни один из инструментов верификации РС не играет доминирующую роль, используются как методики аналитического анализа, так динамического – тестирование.

Инструментов и методик, которые позволяют провести строгую аналитическую верификацию реальных РС пока нет. Аналитические подходы, включая аналитическую проверку моделей, применяются только для упрощенных моделей или для крайне простых реализаций. Достаточно общих подходов, предлагающих синтетические решения, которые совместно используют аналитические и динамические методы верификации пока нет.

Общей проблемой является задача комплексного рассмотрения различных характеристик РС, в первую очередь: функциональных, темпоральных аспектов и аспектов безопасности.

Проблемой остаются масштабируемость и сопоставление свойств поведения исполнимых моделей между собой и между моделями и реализацией. Эти две проблемы, в сущности, определяют границы применимости исполнимых методик.

Литература

[AGEDIS] AGEDIS Consortium, http://www.agedis.de/

[Alu98] R. Alur, T.A. Henzinger, F.Y.C. Mang, S. Qadeer, S.K. Rajamani, and S. Tasiran. Mocha: Modularity in model checking. Proc. of the 10th International Conference on Computer-aided Verification (CAV 1998), Lecture Notes in Computer Science 1427, Springer-Verlag, 1998, pp. 521-525.

[ANSI/IEEE Standard 729-1983] Glossary of Software Engineering Terminology, ANSI/IEEE Standard 729-1983, IEEE Standard, IEEE, NY (1983).

[Artis] http://www.artis-software.com

[Art03] C. Artho, D. Drusinsky, A. Goldeberg, K. Havelund, M. Lowry, C. Pasareanu, G. Rosu, W. Visser. Experiments with Test Case Generation and Runtime Analysis, 10th International Workshop on Abstract State Machines, Sicily, Italy, 2003.

[ASM] http://www.eecs.umich.edu/gasm/

[AsmL] Microsoft Research, http://research.microsoft.com/foundations/AsmL/

[Bau03] M. Bauderon, and M. Mosbah. A Unified Framework for Designing, Implementing and Visualizing Distributed Algorithms. LaBRI, Electronic Notes in Theoretical Computer Science 72 No. 3 (2003).

[Ber96] M. Bernardo, R. Gorrieri. Extended Markovian Process Algebra. Proc. of the 7th Int. Conf. on Concurrency Theory (CONCUR '96), U. Montanari and V. Sassone editors, LNCS 1119:315-330, Pisa (Italy), August 1996.

[Ber98] M.Bernardo, W.R.Cleaveland, S.T.Sims, and W.J.Stewart. TwoTowers: A Tool Integrating Functional and Performance Analysis of Concurrent Systems. Proc. FORTE '98 ,I FIP, North-Holland, 1998.

[BGNSTV03] M. Barnett, W. Grieskamp, L. Nachmanson, W. Schulte, N. Tillmann and M. Veanes. Towards a Tool Environment for Model-Based Testing with AsmL. 3rd International Workshop on FORMAL APPROACHES TO TESTING OF SOFTWARE (FATES 2003), Montréal, Québec, Canada, October 6th, 2003.

[BK85] J.A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. Theoretical Computer Science, 37:77-121, 1985.

[Boz01] M. Bozga, S. Graf, and L. Mounier. Automated validation of distributed software using the IF environment. In S.D. Stoller and W. Visser, editors,

Workshop on Software Model-Checking, associated with CAV'01 (Paris, France) July 2001.

[CADP] http://www.inrialpes.fr/vasy/cadp.html

[Che90] L. Chen, S. Anderson and F. Moller, A Timed Calculus of Communication Systems, LFCS report ECS-LFCS-90-127.

[Compu] http://www.compuware.com/

[Conformiq] Conformiq Software Ltd., http://www.conformiq.com/

[Dai03] Y.S. Dai, M. Xie, K.L. Poh and G.Q. Liu. A Study of Service Reliability and Availability for Distributed Systems. Reliability Engineering and System Safety 79(1):103-112, 2003.

[daimi] http://www.daimi.au.dk/PetriNets/

[Daw96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In Hybrid Systems III, volume 1066 of Lecture Notes in Computer Science. Springer-Verlag, 1996. August 1996.

[doc.ic.ac]http://www.doc.ic.ac.uk/~jnm/book/

[Dru85] D. Drusinsky. The Temporal Rover and the ATG Rover. Springer-Verlag Lecture Notes in Computer Science, 1885, p. 323-329.

[DrS03] D. Drusinsky and M. Shing. Verification of Timing Properties in Rapid System Prototyping, Proc. Rapid System Prototyping Conference 2003 (RSP'2003).

[Dru03] D. Drusinsky and K. Havelund, Execution-Based Real-Time and Time Series Model Checking, Workshop on Model-Checking for Dependable Software-Intensive Systems (International Conference on Dependable Systems and Networks, San Francisco, 2003).

[Drus03] D. Drusinsky, Real-time,On-line,Low Impact, Temporal Pattern Matching, 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003).

[E-LOTOS] http://www.dit.upm.es/~lotos/elotos.html

[Eme90] E.A. Emerson. Temporal and model logic. Handbook of Theoretical Computer Science, Chap 16, edited by J. van Leeuwen, Elesevier Science Publishers, 1990

[estelle] http://www.eecis.udel.edu/~amer/PEL/estelle/

[Fer97] J.Cl. Fernandez, C. Jard, T. J'eron, and C. Viho. An Experiment in Automatic 3 Generation of Test Suites for Protocols with Verification Technology. Science E-10(4), of Computer Programming, 29, 1997.

[Gar02] Hubert Garavel, Frédéric Lang, Radu Mateescu**.** An overview of CADP 2001. European Association for Software Science and Technology (EASST) Newsletter volume 4, pages 13-24, August 2002.

[GH94] S. Gilmore and J. Hillston. The PEPA workbench: a tool to support a process algebra-based approach to performance modelling. In Computer Performance Evaluation, Modeling Techniques and Tools, LNCS 794: 353-368, Springer-Verlag, 1994.

[Gla89] R.J. van Glabbeek and U. Goltz, Equivalence notions for concurrent systems and refinement of actions, Proceedings Mathematical Foundations of Computer Science 1989 (MFCS).

[God97] P. Godefroid. VeriSoft: A Tool for the Automatic Analysis of Concurrent Reactive Software. Proc. 9th Conf. Computer Aided Verification, June 1997.

[God98] P. Godefroid, R.S. Hammer, and L. Jagadeesan. Systematic Software Testing using Verisoft. Bell Labs Technical J., vol. 3, no.2, Apr-June 1998.

[Gun03] Dan Gunter, Brian Tierney: NetLogger: A Toolkit for Distributed System Performance Tuning and Debugging. Integrated Network Management 2003: 97-100.

[Hea94] M.T. Heath. Performance Visualization with ParaGraph. Proc. Second Workshop on Environments and Tools for Parallel Sci. Comput., SIAM, Philadelphia, 1994, pp. 221-230.

[HecLoh03] R. Heckel, and M. Lohmann. Towards Model-Driven Testing. Electronic Notes in Theoretical Computer Science 82 No. 6 (2003).

[Hen97] T.A. Henzinger, P.H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. International Journal on Software Tools for Technology Transfer, 1(1/2):110 – 122, 1997.

[Her00] H.Hermanns,U.Herzog,U.Klehmet,V.Mertsiotakis, and M.Siegle. Compositional performance modelling with the TIPPtool.Perf.Eval., 39(1-4):5 –35, January 2000.

[Hil96] J. Hillston. A Compositional Approach to Performance Modelling. Cambridge University Press, 1996.

[HKN03] H. Hermanns, M. Kwiatkowska, G. Norman, D. Parker and M. Siegle. On the use of MTBDDs for Performability Analysis and Verification of Stochastic Systems, Journal of Logic and Algebraic Programming: Special Issue on Probabilistic Techniques for the Design and Analysis of Systems, 56:23-67, June 2003

[Hoa85] C.A.R. Hoare. Communicating Sequential Processes. Prentice Hall, New York, 1985.

[Hol94] G.J. Holzmann. An Improvement in Formal Verification. Proc. FORTE 1994 Conference, Bern, Switzerland.

[Hol96] G.J. Holzmann, D. Peled, and M. Yannakakis. On nested depth-first search. The Spin Verification System. pp. 23-32 American Mathematical Society, 1996. (Proc. of the 2nd Spin Workshop.).

[Hol97] G.J. Holzmann. The Model Checker SPIN. IEEE Transactions on Software Engineering, Vol. 23, No. 5, pp. 279-295, May 1997.

[Hol98] G.J. Holzmann. An analysis of bitstate hashing. Formal Methods in Systems Design, Nov. 1998.

[Hol02] G.J. Holzmann and M.H. Smith. An automated verification method for distributed systems software based on model checking. IEEE Transactions on Software Engineering, vol. 28, no. 4, April 2002.

[IEEE Std 610.12-1990] IEEE-STD 610.12-1990 – Standard Glossary of Software Engineering Terminology, 1990

[ISO88] ISO/IEC. LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. Technical Report 8807, International Organization for Standardization | Information Processing Systems | Open Systems Interconnection, Geneve, 1988.

[ISO/IEC 8807] http://www.iso.ch/cate/d16258.html

[ITU99] ITU-T. Recommendation Z.100. Specification and Description Language (SDL). Technical Report Z-100, International Telecommunication Union ,Standardization Sector, Geneve, November 1999.

[Klo01] J. Klose, W. Damm. Verification of a Radio-Based Signaling System Using the STATEMATE Verification Environment. Formal Methods in System Design.... 19(2), 2001.

[Lar97] K.G. Larsen, P. Petterson, and W. Yi. UPPAAL: Status & Developments. In O. Grumberg. Editor, Proceedings of CAV'97 (Haifa, Israel), volume 1254 of LNCS, pp. 456-459, Springer, June 1997.

[McM93] K.L McMillan. Symbolic Model Checking. Kluwer Academic Press, 1993.

[Met00] Yves Metivier. Graph Relabelling Systems – A Tool for Encoding, Proving and Studying Distributed Algorithms/ Proceedings of the French Workshop FAC'2000 (Formalization des Activities Concurrents), Toulouse 2000. http://www.irit.fr/FAC2000

[Met99] I. Litovsky, Y. M´etivier, and E. Sopena. Graph relabelling systems and distributed algorithms. In H. Ehrig, H.J. Kreowski, U. Montanari, and G. Rozenberg, editors, Handbook of graph grammars and computing by graph transformation, volume 3, pages 1–56. World Scientific, 1999.

[MDA] OMG Model Driven Architecture

[Mil80] R Milner. A Calculus of Communication System. LNCS 92, 1980.

[Mil89] R Milner. Communication and Concurrency. Prentice Hall, New York, 1989.

[OGEMF02] Open Group, Enterprise Management Forum. 2002, http://www.opengroup.org/management/arm.htm

[OMG99] OMG. Unified Modeling Language Specification. Technical Report OMG UML v1.3 99-06-09, Object Management Group, June 1999.

[Ros94] A.W. Roscoe. Model-Checking CSP. In A Classical Mind, Essays in Honour of C.A.R. Hoare. Prentice-Hall, 1994.

[sdl-forum] http://www.sdl-forum.org/

[Sta92] J. T. Stasko. Animating Algorithms with XTango. SIGAST News, Vol.23 No 2, pp 67-71, 1992.

[Sta93] J. T. Stasko, E. Kraemer. A Methodology for Building Application-Specific Visualizations of Parallel Programs. Journal of Parallel and Distributed Computing, Vol.18 No 2, pp 258-264, 1993.

[Sta95] J. T. Stasko. The PARADE Environment for Visualizing Parallel Program Execution – A Progress Report. Technical Report GA-GIT-GVU-95-03, The Graphics Visualization and Usability Center, Georgia Institute of Technology, Atlanta, 1995.

[stanford.edu] http://theory.stanford.edu/~rvg/abstraction/node2.html

[stanford.edu1] http://theory.stanford.edu/~rvg/research.html#branching

[stanford.edu2] http://theory.stanford.edu/~rvg/abstracts.html#10

[Simulink] http://www.mathworks.com/products/simulink/

[SPIN] http://spinroot.com/spin/whatispin.html]

[Stateflow] http://www.mathworks.com/products/stateflow/

[tausdl] http://www.telelogic.com/products/tau/sdl/index.cfm

[time-rover] http://www.time-rover.com/

[TIPP] http://www7.informatik.uni-erlangen.de/tipp/

[TPTP] Eclipse, http://www.eclipse.org/tptp/index.html

[Var86] Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to automatic program verification. Proc. First IEEE Symp. on Logic in Computer Science, 1986, pp. 322-331.

[Verilog] Verilog. ObjectGEODE Reference Manual. http://www.verilogusa.com/ .

[Wel02] Wells, L.:Performance Analysis Using Coloured Petri Nets. In A. Boukerche, S.K. Das,and S. Majumdar (Eds.):Proceedings of the 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'02).October 11-16, Ft. Worth, TX, USA. IEEE Computer Society: pp 217-221, 2002.

[You02] H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In Ed Brinksma and Kim Guldstrand Larsen, editors, Proceedings of the 14th International Conference on Computer Aided Verification, volume 2404 of Lecture Notes in Computer Science, pages 223-235, Copenhagen, Denmark, July 2002. Springer.