Журнал "Открытые системы", #12, 2003 год // Издательство "Открытые системы" (www.osp.ru)
Постоянный адрес статьи: http://www.osp.ru/os/2003/12/045.htm

Верификация программ с помощью моделей

Александр Аграновский, Вадим Зайцев, Борис Телеснин, Роман Хади

18.12.2003

Чем критичнее для бизнеса программа, тем дороже обходятся дефекты в ней. Но, к сожалению, создание априори безошибочных программ дело чрезвычайно нетривиальное — если не сказать, невозможное. На практике часто используются методы валидации и верификации, т.е. проверки программного обеспечения на корректность реализации поставленной задачи путем сравнения с требуемыми свойствами.

Согласно модели T.I.N.A. (*www.tinac.com* — «Открытая архитектура для разработки распределенного программного обеспечения»), *верификация* продолжается вплоть до момента кодирования программы, а валидация осуществляется непосредственно после. Однако в большинстве случаев процессы верификации, валидации, тестирования и реализации пересекаются по времени.

Сегодня используются два подхода к валидации программного обеспечения. Первый подход, *дедуктивный*, представлен такими направлениями исследований, как автоматическое доказательство теорем, использованием мультимножеств и графов, а также разнообразных специализированных алгебр. Программная система описывается в рамках некоего формализма, после чего выполняется строгое математическое доказательство обладания данной системой тех или иных свойств. Второй подход — *модельный*; его последователи не стремятся вписать систему в рамки теории, а вместо этого строят модель системы, которую можно рассматривать как машину или автомат. Любое требование к системе проверяется для каждого возможного состояния автомата.

Сильные и слабые стороны модельного подхода

Модельных подходов известно, по меньшей мере, несколько дюжин — конечные автоматы, сети Петри, временные автоматы, логическое описание и т.п. Попробуем перечислить присущие им общие сильные свойства.

Модельный подход поддерживает не только полную, но и частичную верификацию, которая может быть направлена на проверку только одного небольшого свойства, абстрагировавшись от менее важных деталей системы. Иными словами, для проведения верификации не обязательно добиваться формализации всех без исключения требований спецификации. В отличие от тестирования и использования симуляторов, в модельном подходе не существует такого понятия, как *вероятность обнаружения ошибки*: если ошибка есть, она будет обнаружена за конечное время.

В том случае, когда свойство оказывается нарушенным, в виде контрпримера предоставляется диагностирующая информация.

Процесс проверки моделей не требует ни ручного управления со стороны пользователя, ни высокого уровня профессионализма. Имея модель, можно автоматически проверять на ней необходимые свойства. Процесс проверки интегрируется в стандартный цикл проектирования, позволяя, как показывает практика, уменьшить время создания приложений с учетом проведения рефакторинга программного кода.

Однако у модельного подхода есть и слабые стороны. Верификация осуществляется по модели, а не по реальной системе, поэтому ценность полученного результата напрямую зависит от корректности модели, что требует высокого уровня подготовки персонала, создающего модели программ.

Преобладает ориентация на приложения, в которых главную роль играет *поток управления*, а не *поток данных*, так как данные имеют тенденцию принимать значения из бесконечных множеств. Такая ориентация уменьшает возможности универсального применения, однако обычно это не столь существенно при разработке больших аппаратно-программных комплексов, поскольку практически все существующие виды модульных приложений, из которых складываются подобные комплексы, можно

либо в том или ином виде привести к модели «потока управления», либо корректировать методику тестирования для каждого конкретного модуля. Модельный подход не может эффективно применяться без точных алгоритмов принятия решений. Нет гарантий полноты: проверяются только те свойства, которые указаны явно.

Построение моделей и формулировка требований требуют высокого уровня знаний и умения их применять. Результаты могут вводить в заблуждение (верификатор — тоже программа и тоже может ошибаться, модель может содержать ошибку и т.п.; правда, основные процедуры проверки моделей формально доказаны с помощью пакетов автоматического доказательства теорем). Нет верификаторов, поддерживающих обобщения, например, нельзя проверить систему, если в ней не зафиксировать число сущностей.

Примеры успешного применения модельного подхода можно обнаружить, изучая процесс разработки сложных систем, оперирующих большими потоками данных: СУБД, комплексы потоковой обработки речевой и текстовой информации, системы обеспечения информационной безопасности. Модельный подход к верификации программного обеспечения позволяет, при правильном разбиении всего комплекса, при проектировании и разработке модулей и более атомарных составляющих выявлять логические ошибки еще на этапе проектирования. Так, при разработке программного обеспечения потоковой обработки растровых изображений в рамках модельного подхода была сформирована модель для верификации менеджера заданий для потоковой обработки и обработчиков атомарных заданий, позволившая выявить ошибки в проектировании протоколов взаимодействия модулей комплекса и алгоритме определения обработчика атомарного задания. Данная модель основана на использовании сетей Петри и сопутствующих алгоритмов [2, 4].

Качественные характеристики

Еще не так давно все требования к приложениям делились на функциональные и нефункциональные. Первые, как правило, были представлены двоичным значением «работает/не работает», а вторые — длинным списком свойств, верифицируемых субъективно (например, «дружелюбность, устойчивость, безопасность»). В последнее время ситуация изменилась, и полный список типов возможных требований был стандартизован в рамках стандарта ISO 9126 [1].

Говоря о функциональности, обычно подразумевают некоторое множество атрибутов, рассчитанных на существование определенного набора функций и их специальных свойств, достигающих поставленных целей [1, 3].

- **Пригодность.** - Выполняет ли приложение предназначенную ему задачу? Может быть верифицировано путем моделирования правильного сопутствующего окружения (подход, аналогичный тестированию).
- **Точность.** - Насколько точны результаты работы приложения? Трудно реализуется при модельном подходе; логическая верификация в данном случае будет более эффективна.
- **Безопасность.** - Не происходит ли неавторизованной утечки информации? Верифицируется напрямую с формулированием соответствующих запросов. Также существует целый ряд немодельных верификаторов, решающих эту же задачу.
- **Соответствие.** - Соответствует ли реализованная функция данному стандарту? Стандарт используется как спецификация (источник требований), реализация функции моделируется.
- **Совместимость.** - Может ли данное приложение общаться с соответствующими программными продуктами от других производителей? Близким приближением является подразумеваемая совместимость при наличии соответствия стандарту и отсутствии недокументированных возможностей. При необходимости более точной проверки выполняет автоматическое дизассемблирование и эмуляцию заданных участков программного кода, ручную отладку, построение графа передачи управления и данных.

Множество атрибутов надежности характеризуют способность программного обеспечения поддерживать определенный уровень предоставляемых услуг при данных условиях и в течение заданного промежутка времени [1, 5].

- **Завершенность.** - Является ли изначально предоставляемый уровень услуг достаточным? Все ли было реализовано? Это свойство по определению не может быть проверено формальным тестированием: на каждую ожидаемую функцию формулируется требование (или множество требований), которые проверяются на модели.
- **Устойчивость к ошибкам.** - Ведет ли себя программа адекватно в случае предоставления

заведомо неверных входных данных? Очень неэффективно и громоздко реализуется в модельном подходе, существуют неплохие методы тестирования, решающие эту проблему.
- **Устойчивость к окружению (прочность).** - Может ли приложение работать нормально в нестандартном или неустойчивом окружении? Применение модельного подхода в данном случае возможно только при наличии возможности моделирования окружения. Однако корректное моделирование стресс-ситуации - весьма нетривиальная задача.
- **Восстанавливаемость.** - Может ли приложение продолжать работу после сбоя? Как правило, это свойство явно прописывается в программе и нуждается только в проверке. Может быть проверено как модельной верификацией, так и тестированием.

Множество атрибутов по удобству пользования характеризует трудности при использовании программного обеспечения и их субъективную оценку тем или иным множеством пользователей [1, 6].

- **Понятность.** - Насколько интуитивно ясен пользовательский интерфейс приложения? Не поддается научной формализации, несмотря на то, что менее формальные правила существуют уже давно, модельная верификация невозможна.
- **Обучаемость.** - Приспосабливается ли приложение к специфике пользователя? Используются алгоритмы искусственного интеллекта, которые могут быть верифицированы, соответственно, может быть верифицирован и признак.
- **Управляемость.** - Легко ли управлять работой приложения? Эта область, традиционная для бета-тестирования, в последнее время переходит в руки специалистов по пользовательским интерфейсам.

Множество атрибутов производительности выявляет связь уровня предоставляемых приложением услуг с объемом используемых при этом ресурсов [1, 7].

- **Поведение во времени.** - Адекватен ли временной график использования ресурсов? В данном случае нужно тестировать реальную систему, а не ее модель (например, для нахождения утечки памяти). Абсолютно не подходит для модельной верификации.
- **Использование ресурсов.** - Эффективно ли используются ресурсы? Имеется направленность на реальную систему и существуют эффективные методы формального тестирования, которые в основном базируются на смеси сетй Петри и специализированных языках описания моделей верификации, при прогонке которых происходит количественная оценка потенциально используемых ресурсов; максимальное значение дает вполне эффективную оценку, пригодную для большинства реализаций.
- **Алгоритмизация.** - Насколько оптимальны использованные алгоритмы? Классический анализ алгоритмов вместе с формальной их верификацией дает быстрые и точные результаты.

Множество атрибутов поддержки связано с усилиями по внесению определенных изменений в работающее приложение [1, 8].

- **Анализируемость.** - Насколько легко определить части, нуждающиеся в изменении? Не поддается формализации.
- **Изменяемость.** - Какие усилия требуются для внесения изменений? Не поддается формализации, уровень может быть установлен априори.
- **Настраиваемость.** - Можно ли достичь желаемого эффекта без изменения самой программы, изменяя только настройки? Задача решается тестированием в реальных условиях.
- **Стабильность.** - Как ведет себя программа при внесении изменений на лету? Эффективно решается модельной верификацией с помощью недетерминированных параллельных процессов.
- **Тестируемость.** - Насколько легко проверяется работа изменившегося контура? Решается параллельно с тестированием или превентивно явным образом и к верификации отношения практически не имеет.

Множество атрибутов переместимости характеризует способность программного обеспечения быть перенесенным из одного окружения в другое [1, 9].

- **Приспособляемость.** - Может ли приложение изменяться в соответствии с изменениями окружения? Взаимодействующие недетерминированные последовательные процессы дают хороший результат, в том числе, и в модельном подходе.
- **Устанавливаемость.** - Может ли приложение устанавливаться на разные платформы или в разные конфигурации? Как правило, явно задается в спецификации и явно реализуется и в проверке не нуждается.

- **Согласованность.** - Какие стандарты были использованы в приложении? Не нуждается в проверке, однако само соответствие стандартам проверять можно и нужно.
- **Заменяемость.** - Может ли приложение быть использовано так же, как его эквивалент от другого производителя? Зависит ли от списка опций соответствующих приложений, которые могли бы быть или должны были быть реализованы? Это относится к фазе формулирования требований, поэтому в верификации не участвует.

Мы привели общий список свойств, которые могут быть проверены с помощью техник модельной верификации и валидации, сделав его насколько возможно кратким. Свойства, не упомянутые (например, масштабируемость или живучесть), но встречающиеся на практике, могут быть сведены к данному списку.

Литература

1. International Standard ISO/IEC 9126. Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for their Use. International Organization for Standardization, International Electrotechnical Commission, Geneva, 1991.
2. Chamillard A.T., Clarke L.A. Improving the Accuracy of Petri Net-based Analysis of Concurrent Programs, Proceedings of the International Symposium on Software Testing and Analysis, San Diego, 1996.
3. Huget M.-F., Wooldridge M. Model Checking for ACL Compliance Verification, Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, 2003.
4. Vaishnavi V.K., Fraser M.D. A Validation Framework for a Maturity Measurement Model for Safety-critical Software Systems, Proceedings of the 36th Annual Southeast Regional Conference, 1998.
5. Kumar S., Li K. Using Model Checking to Debug Device Firmware, ACM SIGOPS Operating Systems Review, 36(SI), Winter 2002.
6. Baresi L., Heckel R., Thone S., Varro D. Modeling and Validation of Service-oriented Architectures: Application vs. Style, Proceedings of the 9th European Software Engineering Conference, 2003.
7. Aagard M.D., Jones R.B., Kaivola R., Kohatsu K.R., Seger C.-J.H. Formal Verification of Iterative Algorithms in Microprocessors, Proceedings of the 37th Conference on Design Automation, 2000.
8. Bhargavan K., Obradovic D., Gunter C.A. Formal Verification of Standards for Distance Vector Routing Protocols, Journal of the ACM, 49(4), July 2002.
9. Ramalingam G., Warshavsky A., Field J., Goyal D., Sagiv M. Deriving Specialized Program Analyses for Certifying Component-client Conformance, Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, 2002.

Александр Аграновский — директор, Борис Телеснин, Вадим Зайцев — научные сотрудники, Роман Хади (rhady@rsu.ru) — заведующий лабораторией ГНУ НИИ «Спецвузавтоматика» (Ростов-на-Дону).

Модельная верификация в реальном проекте

Для создаваемой нашим предприятием системы обнаружения и защиты от сетевых компьютерных атак были разработаны протоколы взаимодействия различных распределенных компонентов: информационные зонды, собирающие информацию о сетевом трафике; центры сбора и обработки информации, поступающей от зондов; мобильная консоль администратора. Предложенный протокол на алгоритмическом уровне был проанализирован с помощью BAN-логики — аналог модельного верификатора для алгоритмов на языках высокого уровня. Программная реализация анализировалась с помощью языка PROMELA и инструментального верификатора SPIN (spinroot.com/spin).

С помощью BAN-логики была получена модель, детально описывающая, кто и какой информацией владеет при обмене данными между различными компонентами системы. Это позволило выявить ошибки в алгоритме аутентификации и методе использования криптографических средств защиты информации. С помощью языка PROMELA была сформулирована верификационная модель алгоритма, которая затем была применена к реализации алгоритма взаимодействия на языке программирования Си (компилятор gcc и операционная система FreeBSD 5.0). Это, в свою очередь, позволило выявить факты «утечки» памяти, которые приводили к полному останову программы при определенных, весьма узких, входных условиях. При этом используемый уровень абстракции модели (по сути, подход MDA) позволил использовать результаты верификации на всех платформах, для которых разрабатывалось приложение (Windows, FreeBSD, Linux, МСВС).

Использование модельного подхода дало очевидное преимущество в самых уязвимых местах с точки зрения всего проекта. Грубые ошибки были удалены еще до начала бета-тестирования, что сказалось на общих сроках выполнения проекта и выпуска промышленных версий.

Журнал "Открытые системы", #12, 2003 год // Издательство "Открытые системы" (www.osp.ru)
Постоянный адрес статьи: http://www.osp.ru/os/2003/12/045.htm