УДК 004.052

Е.И. Политько, О.М. Тарасюк, А.В. Горбенко

Национальный аэрокосмический университет им. Н.Е. Жуковского "ХАИ", Харьков

МЕТОД ФОРМАЛЬНОЙ ВЕРИФИКАЦИИ UMC: UML MODEL CHECKING

В статье анализируются существующие проблемы инженерии программного обеспечения, обусловленные сложностью современных программ и параллельностью выполнения взаимодействующих процессов. Выполнен обзор формальных методов разработки и верификации, позволяющих доказать соответствие моделей программ и систем формальным требованиям. Рассмотрены особенности метода формальной верификации Model Checking и его развитие для верификации диаграмм состояний UML. Рассмотрены синтаксис и семантика темпоральной логики μUCTL и формальное представление моделей UML в виде системы перехода с двойной пометкой L2TS.

Ключевые слова: *программирование, формальные методы, верификация, Model Checking, UML.*

Введение

В настоящее время безотказное и эффективное функционирование экономических и бизнес-процессов предприятий, систем управления технологическими процессами, а также множества технических объектов в различных отраслях промышленности и сферах деятельности человека в значительной степени зависит от качества и надежности программного обеспечения [1]. Однако разработка программного обеспечения остается фактически единственной областью инженерной деятельности, где разработчик не может гарантировать качество своей продукции – реальные программы очень часто содержат ошибки, последствия которых не всегда безобидны.

Например [2], спутник «Экспресс-АМ4», который должен был обеспечить доступ к Интернету и телевещанию на Дальнем Востоке, 25 марта 2012 года был выведен с околоземной орбиты и затоплен, поскольку при его запуске возник сбой в работе программного обеспечения разгонного блока, в результате чего спутник оказался на нерасчетной орбите. В связи с инцидентом страховая компания выплатила страховую сумму в размере более 7,5 млрд. руб. Однако более значимым является тот факт, что широкое внедрение средств автоматизации и компьютеризации снижает безопасность систем повседневного использования и технических объектов, окружающих человека. Интернет-ресурсы полны сообщениями о техногенных авариях (от гибели ребенка в автоматизированном туалете до аварий с человеческими жертвами автомобилей Toyota из-за ошибок в электронной системе торможения), причиной которых является некорректная работа программных систем управления. Такие случаи делают проблему обеспечения корректности и безопасности компьютерных программ особо важной.

Сложность программ давно подошла к границе их понимания человеком, а, следовательно, к границе их управляемости, поэтому на сегодня наблюдается устойчивый рост числа ошибок в разработанных и сданных заказчику программных продуктах. Особенно подвержены ошибкам параллельные, распределенные и многопоточные программы, характерные для систем управления, причем именно та-

кие программы получают сейчас все большее распространение [3]. Например, в современном автомобиле имеется целая сеть из десятков связанных микропроцессоров, а бытовые ноутбуки с многоядерными процессорами ввели параллельное программирование и примитивы синхронизации параллельных процессов в широкий обиход программистов всех уровней. В течение же ближайших лет большинство новых автомобилей будут построены по схеме drive-by-wire [4] – т.е. непосредственные механические соединения между органами управления и исполнительными устройствами (например, рулем и колесами) будут заменены микропроцессорными устройствами.

Таким образом, обеспечение высокой надежности и безошибочности программного обеспечения современных систем управления является одной из главных задач ИТ-индустрии. В связи с этим, актуальным представляется внедрение в практику инженерии программного обеспечения методов формальной верификации [5], дополняющих традиционные методы тестирования и отладки, и позволяющих повысить безотказность и безопасность программ.

Цель статьи – анализ перспективных методов формальной верификации программного обеспечения Model Checking и перспективы их совместного использования с традиционными инструментами объектно-ориентированного анализа, такими как UML.

1. Формальные методы. Model Checking

В настоящее время наблюдается устойчивый интерес в применении формальных методов для автоматического доказательства корректности разрабатываемых систем. В этом направлении используется два основных подхода:

– автоматическое доказательство теорем (Automated Theorem Proving), которое базируется на принципах дедуктивного анализа и аппарата пред- и пост-условий и заключается в использовании набора логических аксиом и правил вывода для доказательства правильности (непротиворечивости) имеющегося описания системы;

– верификация модели (Model Checking), в рамках которого выполняется верификация поведенческих свойств системы на основе полного рассмотрения (обхода) всех возможных состояний, в которых система может оказаться в процессе функционирования.

Model Checking – это набор идей и методов для построения поведенческих моделей работающих систем, протоколов или программ, математической формулировки требований к ним, отражающих правильность их работы, т.е. формальной спецификации свойств, и алгоритмов автоматической проверки (доказательства или опровержения) этих требований (свойств) на всем множестве состояний модели.

Если модель удовлетворяет указанным требованиям, то программа-верификатор сообщает об этом. Если же обнаруживается ошибка, то она предоставляет контрпример, который показывает, при каких условиях могло возникнуть данное несоответствие.

Контрпример представляет собой сценарий, в котором модель ведет себя нежелательным образом и нарушает одно или несколько верифицируемых свойств. Это означает, как правило, что модель ошибочна и подлежит пересмотру. Однако в некоторых случаях это может означать, что неверны формальные требования или же модель системы построена некорректно.

Обратим внимание, что модель программы не всегда полно отражает ее поведение. Разработчик при построении модели, как правило, абстрагируется от несущественных ее свойств. Такая концепция дает возможность уменьшить размер самой модели и ускорить процесс ее проверки.

Проверка модели позволяет разработчику обнаружить ошибку и исправить модель или требования. Если не найдено ни одной ошибки, разработчик может усовершенствовать описание модели (сделать модель более реалистичной, приняв во внимание больший набор свойств), как правило, увеличив ее размер, и перезапустить процесс верификации.

Основная трудность моделирования – не потерять важные детали программы, а трудность задания требований – сформулировать их корректно и исчерпывающе. Для построения модели системы в рамках метода формальной верификации Model Checking используется структура Крипке – автомат с конечным числом состояний, характеризующийся только самими состояниями. Для формулировки требований к системе применяются темпоральные логики (LTL, CTL и др.), позволяющие определить её динамические свойства.

Алгоритмы для Model checking обычно базируются на полном просмотре пространства состояний модели: для каждого состояния проверяется, удовлетворяет ли оно сформулированным требованиям. Алгоритмы гарантированно завершаются, так как число состояний модели конечно.

Общий алгоритм формальной верификации Model Checking представлен на рис. 1.

В качестве недостатка традиционных формальных методов, в том числе и Model Checking, следует указать весьма ограниченную поддержку современных технологий и парадигм программирования. В частности, для описания модели системы традиционным является использование абстрактного языка спецификации алгоритмов Promela.

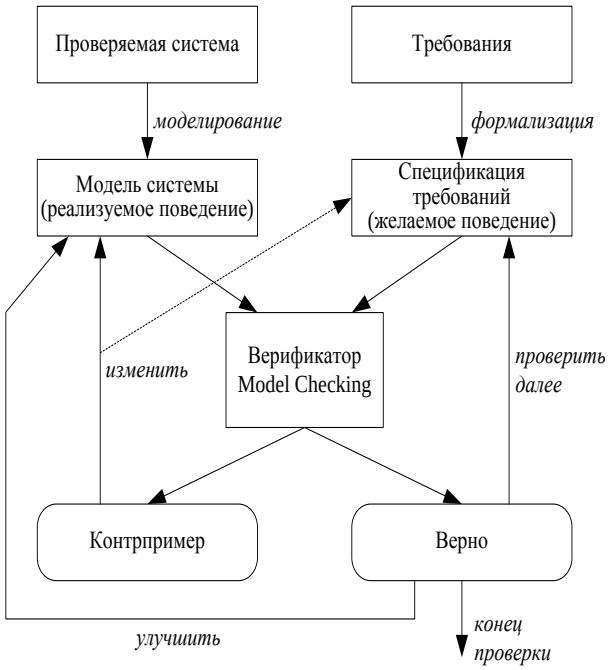

Рис. 1. Алгоритм формальной верификации Model Checking.

Описание на языке Promela затем может быть преобразовано в структуру Крипке. Однако, разработанные на языке Promela модели существенно отличаются от верифицируемых программ, обычно написанных на языках программирования высокого уровня, например, Java или C.

Программы на Promela не имеют классов, не используют указателей и не поддерживают сложные структуры данных. Они представляют плоскую структуру взаимодействующих параллельных процессов, имеют минимум управляющих конструкций, а все переменные имеют конечные области определения.

Поэтому на сегодня одним из перспективных направлений развития методов формальной разработки и верификации программ является поддержка объектно-ориентированной модели взаимодействия и средств визуального моделирования. В связи с этим можно выделить метод формальной верификации UMC (UML Model Checking), который в качестве исходной модели системы использует набор диаграмм состояний (State chart diagrams) взаимодействующих объектов, разработанных с использованием спецификации унифицированного языка моделирования UML 2.0.

2. Метод формальной верификации UMC

UMC [6] – метод и инструментальное средство для формальной верификации спецификаций, описанных с помощью темпоральной логики μUCTL, для проверки корректности UML-диаграмм состояний (см. рис. 2).

UMC принимает в качестве входных данных набор описаний диаграмм состояний (описывающих динамическое поведение системы), диаграмму объектов, описывающую начальную конфигурацию системы, набор «наблюдаемых критериев», который включает атрибуты объекта и события связи, которые мы хотим пронаблюдать, а также μUCTL-формулу, представляющую свойство, которое необходимо верифицировать.

Формула μUCTL проверяется «на лету», инкрементно генерируя систему состояний и переходов L2TS.

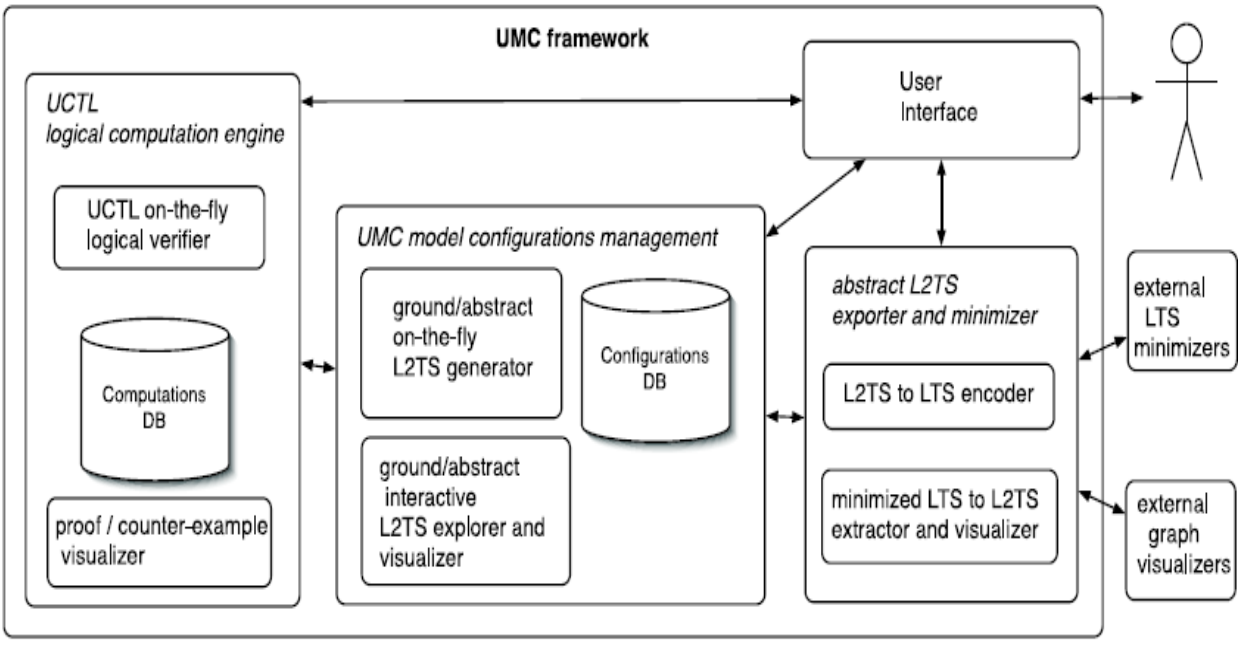

Рис. 2. Структура метода UMC

2.1. Формальное моделирование системы с помощью диаграмм состояний UML

UML (Unified Modeling Language) представляет собой унифицированный язык графического моделирования объектно-ориентированных систем [7]. UML поддерживает множество диаграмм различного типа и назначения, среди которых – диаграммы состояний, предназначенные для описания динамических свойств систем. UML позволяет ассоциировать диаграмму состояний с каждым объектом, составляющим систему, а также определить семантику диаграммы состояний с точки зрения автоматов. Все возможные поведения системы могут быть описаны как возможные эволюции набора взаимосвязанных автоматов – диаграмм состояний UML, которые могут быть формально представлены в виде системы перехода с двойной пометкой L2TS (Doubly Labeled Transition Systems) [8] в которых состояния представляют варьирующиеся системные конфигурации и границы возможных эволюций системной конфигурации.

LTS – это четверка (Q, q0, Evt, R), где Q – это множество состояний; $q0 \in Q$ – начальное состояние; Evt – конечный набор наблюдаемых событий; $R \subseteq Q \times 2Evt \times Q$ – отношение переходов.

В свою очередь L2TS – это шестерка (Q, q0, Evt, R, AP, L), где (Q, q0, Evt, R) – то же самое, что и в LTS; AP – набор атомарных предложений; $L : Q \rightarrow 2AP$ – функция, помечающая каждое состояние подмножеством AP. Другими словами L2TS – это, так называемая, «структура Крипке с переходами», которая является расширением структуры Крипке путем пометки переходов.

2.2. Темпоральная логика μUCTL

Большинство языков спецификации, используемых для описания динамического поведения систем, являются либо состояние-ориентированными, либо событийно-ориентированными. В первом случае описание системы строится вокруг внутренних свойств состояний; во втором случае описание строится вокруг происходимых событий, возникающих при переходе из одного состояния в другое.

Темпоральные логики являются широко признанным и полезным формализмом для выражения свойств живучести и безопасности комплексных систем. Большинство обычно используемых темпоральных логик, такие как CTL или LTL, базируются только на одной из парадигм (состояния/события).

Метод формальной верификации UMC для спецификации требуемых свойств UML и L2TS моделей использует состояние/событийно-ориентированную темпоральную логику μUCTL [6].

Формула μUCTL строится над состояниями Evt.

Семантика событийной формулы. Удовлетворяющее соотношение (|=) для событийной формулы формы $\eta \models \chi$ определено над множеством событий как указано ниже:

$\eta \models tt$ (выполняется всегда);

$\eta \models e$ iff $e \in \eta$;

$\eta \models \tau$ iff $\eta = \varnothing$;

$\eta \models \neg\chi$ iff not $\eta \models \chi$;

$\eta \models \chi \wedge \chi'$ iff $\eta \models \chi$ and $\eta \models \chi'$.

Семантика событийной формулы требует, чтобы событию e соответствовало строго одно событие в η.

Синтаксис μUCTL определен следующим образом:

– формула состояния $\varphi ::= true \mid p \mid \neg\varphi \mid \varphi \wedge \varphi' \mid E\,\pi \mid A\,\pi$;

– формула пути) $\pi ::= X\chi\varphi \mid \varphi\, \chi U \chi' \varphi' \mid \varphi\, \chi W \chi' \varphi$.

Допустим, что (Q, q0, Evt, R, AP, L) – это L2TS, $q \in Q$ и $\sigma \in fpath(q')$ для некоторого $q' \in Q$. Удовлетворяющее соотношение для μUCTL формулы описано ниже:

$q \models true$ =holds always;

$q \models p$ iff $p \in L(q)$;

$q \models \neg\varphi$ iff not $q \models \varphi$;

$q \models \varphi \wedge \varphi'$ iff $q \models \varphi$ and $q \models \varphi'$;

$q \models E\,\pi$ iff $\exists\, \sigma \in fpat=(q) : \sigma \models \pi$;

$q \models A\pi$ iff $\forall\, \sigma \in fpat=(q) : \sigma \models \pi$;

$\sigma \models X\chi\varphi$ iff $\sigma\{1\} \models \chi$ and $\sigma(2) \models \varphi$;

$\sigma \models \varphi\, \chi U \chi' \varphi'$ iff $\exists\, j \geq 1$: $\sigma(j) \models \varphi$, $\sigma\{j\} \models \chi'$ and $\sigma(j+1) \models \varphi'$ and $\forall\, 1 \leq i < j : \sigma(i) \models \varphi$ and $\sigma\{i\} \models \chi$;

$\sigma \models \varphi\, \chi W \chi' \varphi'$ iff either $\sigma \models \varphi\, \chi U \chi' \varphi'$ or $\forall\, j \geq 1$: $\sigma(j) \models \varphi$ and $\sigma\{j\} \models \chi$.

При верификации UML моделей может возникнуть проблема бесконечности пространства состояний. Для решения этой проблемы в UMC можно задавать лимит глубины поиска в генерации модели. В случае если верный результат не был найден, то увеличивается лимит глубины генерации модели и процесс начинается заново.

Заключение

Методы формальной верификации программного обеспечения компьютерных систем управления позволяют гарантировать проверку выполнения моделью системы верифицируемых свойств. В настоящее время эти методы активно развиваются в направлении снижения общей стоимости формальной проверки, поддержки современных концепций программирования и минимизации «ручного» труда при переходе от модели системы к её реализации.

Среди перспективных направлений развития метода формальной верификации Model Checking следует выделить метод UMC, который позволяет работать с моделями, описывающими одновременно как состояния, так и переходы системы. В качестве модели системы UMC принимает диаграммы состояний UML, что является привлекательным для его внедрения в современную практику разработки программного обеспечения.

Диаграммы состояний UML-модели определяются одновременно выполняющимися UML-автоматами. Эти автоматы имеют формальное представление и описывают динамические аспекты поведения компонентов систем, освещая и состояние-ориентированный, так и событийно-ориентированный аспекты работы разрабатываемый системы. Оба аспекта хорошо представлены в L2TS (doubly labelled transition systems), являющимися расширением LTS. Основной характеристикой L2TS систем является то, что их состояния и переходы помечены набором предикатов и набором событий, соответственно.

Применение UMC для верификации моделей UML на ранних этапах жизненного цикла разработки приближает его к методам формальной спецификации и позволяет сократить стоимость исправления выявленных дефектов.

Список литературы

1. Боярцева Е.С. Надежность программного обеспечения автоматизированных систем обработки информации [Электронный ресурс] / Е.С. Боярцева // Молодежная электронная научная школа-конференция «Актуальные проблемы защиты информации и информационной безопасности». – Режим доступа к ресурсу: http://stavkombez.ru/ conf/2012/03/29/надежность-программного-обеспечения.

2. Спутник "Экспресс-АМ4" упал в Тихий океан [Электронный ресурс]. – Режим доступа к ресурсу: http://lenta.ru/news/2012/03/25/am4/.

3. Власенко А.Ю. Архитектура и принципы работы масштабируемого инструментального средства динамического обнаружения семантических ошибок в MPI-программах [Текст] / А.Ю. Власенко // Тр. Всероссийской конф. «Высокопроизводительные параллельные вычисления на кластерных системах». – Владимир (Россия), 2009. – С. 73.

4. Bernardini D. Drive-by-wire Vehicle Stabilization and Yaw Regulation: a Hybrid Model Predictive Control Design [Text] / D. Bernardini, S. Di Cairano, A. Bemporad, H.E. Tseng // Proc. Joint 48th IEEE Conference on Decision and Control and 28th Chinese Control Conference. – Shanghai (China), 2009. – P. 7621-7626.

5. Буренков В.С. Проблемы формальной верификации технических систем [Электронный ресурс] / В.С. Буренков, С.Р. Иванов, А.Я. Савельев // Молодежный научно-технический вестник. – 2012. – №4. – Режим доступа к ресурсу: http://technomag.edu.ru/doc/373672.html.

6. Gnesi S. A Molel Checking Verification Environment For UML Statecharts [Text] / S. Gnesi, F. Mazzanti // Proc. XLIII Congresso Annuale (AICA'2005). – Udine (Italy), 2005. – P. 7-16.

7. Ларман К. Применение UML 2.0 и шаблонов проектирования [Текст] / К. Ларман. – М.: Вильямс, 2006. – 736 с.

8. Rigorous Software Engineering for Service-Oriented Systems. LNCS 6582. [Text] / M. Wirsing, M. Hölzl (Eds.). – Berlin: Springer, 2011. – 737 p.

Поступила в редколлегию 26.12.2012

Рецензент: д-р техн. наук, проф. В.С. Харченко, Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ», Харьков.

МЕТОД ФОРМАЛЬНОЇ ВЕРИФІКАЦІЇ UMC: UML MODEL CHECKING

Є.І. Політько, О.М. Тарасюк, А.В. Горбенко

У статті аналізуються існуючі проблеми інженерії програмного забезпечення, обумовлені складністю сучасних програм і паралельністю виконання взаємодіючих процесів. Виконано огляд формальних методів розробки і верифікації, що дозволяють довести відповідність моделей програм і систем формальним вимогам. Розглянуто особливості методу формальної верифікації Model Checking і його розвиток для верифікації діаграм станів UML. Розглянуто синтаксис і семантику темпоральної логіки μUCTL і формальне подання моделей UML у вигляді системи переходу з подвійною позначкою L2TS.

Ключові слова: *програмування, формальні методи, верифікація, Model Checking, UML.*

UMC METHOD OF FORMAL VERIFICATION: UML MODEL CHECKING

E.I. Politko, O.M. Tarasyuk, A.V. Gorbenko

The paper analyzes the existing problems of software engineering, due to the complexity of modern programs and parallel execution of cooperating processes. A review of formal methods for the design and verification, allowing proving the conformity of program models and system's formal requirements, was performed. The features of the method of formal verification Model Checking and its development for the verification of UML state-charts were reviewed. The syntax and semantics of temporal logic μUCTL and formal representation UML models, as doubly labeled systems (L2TS), were investigated.

Keywords: *programming, formal methods, verification, Model Checking, UML.*