УДК 004.052

О.М. ТАРАСЮК, А.В. ГОРБЕНКО, В.С. ХАРЧЕНКО

НАСКОЛЬКО ФОРМАЛЬНЫ ФОРМАЛЬНЫЕ МЕТОДЫ?

Анотація. *У статті аналізуються проблеми, пов'язані з розвитком та застосуванням формальних методів під час створення й верифікації програмного забезпечення та комп'ютерних систем. Дається класифікація інваріантів. Уточнюються поняття й процедура синтезу системи інваріантів та її оптимізація за певними критеріями. Розглянуто моделі життєвого циклу за умов використання формальних методів Event-B та Model Checking, їх особливості та обмеження. Сформульовано напрямки подальших досліджень у цій області.*
Ключові слова: *формальні методі, система інваріантів, Event-B, Model-Checking.*

Аннотация. *В статье анализируются проблемы, связанные с развитием и внедрением формальных методов при разработке и верификации программного обеспечения и компьютерных систем. Дается классификация инвариантов. Уточняются понятие и процедура синтеза системы инвариантов и ее оптимизации по некоторым критериям. Предлагаются модели жизненного цикла программного обеспечения при использовании формальных методов Event-B и Model Checking. Анализируются их особенности и ограничения. Формулируются направления дальнейших исследований в этой области.*
Ключевые слова: *формальные методы, система инвариантов, Event-B, Model-Checking.*

Abstract. *The paper analyses existing obstacles associated with development and practical application of formal methods in the design and verification of software and computer systems. The classification of invariants is given in the paper. We specify a concept of the system of invariants and discuss a procedure of its synthesis and optimization. The changes in software life cycle caused by applying formal methods Event-B and Model Checking are discussed. The directions of further researches in this area are also given.*
Key words: *formal methods, system of invariants, Event-B, Model-Checking.*

1. Введение. Постановка задачи

1.1. Вызовы размерности и формальные методы

При разработке и верификации сложных компьютерных систем и их программного обеспечения одной из ключевых является проблема размерности. Она обусловлена большим числом возможных состояний разрабатываемой системы, возрастающим количеством вариантов проектных решений, практической невозможностью полного перебора тестовых комбинаций при проверке, огромными объемами документации, которая должна быть проанализирована при верификации и аудите и др. В то же время для критических приложений растет цена проектных ошибок и ошибок верификации, поскольку они увеличивают риски скрытых дефектов, приводящих к нештатному функционированию и/или неадекватной реакции на такое функционирование.

Естественной реакцией на подобную ситуацию является желание уменьшить или полностью исключить неформализованные проектные решения, зависящие от человеческого фактора. Одним из естественных ответов вызовам, связанным с этими факторами, является использование так называемых формальных методов. В широком смысле под ними понимаются методы и соответствующие им технологии, основанные на применении математически обоснованных правил и процедур. Число таких методов и их модификаций чрезвычайно велико. Примерами формальных методов, используемых для анализа и синтеза объектов различной природы, служат методы, основанные на марковских случайных процессах, теории систем массового обслуживания, сетях Петри и др.

В узком смысле формальными называют методы автоматического доказательства корректности разрабатываемых систем. В этом направлении используются два основных подхода:

– автоматическое доказательство теорем (Automated Theorem Proving), которое базируется на дедуктивном анализе и использовании набора логических аксиом и правил вывода для доказательства правильности (непротиворечивости) описания системы [1]; одним из наиболее известных формальных методов здесь является Event-B [2], который интегрирует многошаговую процедуру детализации разрабатываемой системы от спецификации до программного кода с автоматическим доказательством соблюдения ключевых свойств (инвариантов) системы;

– верификация модели (Model Checking), в рамках которой выполняется проверка определенных свойств системы на основе полного рассмотрения (трассирования) всех возможных состояний, в которых система, а точнее её модель, может оказаться в процессе работы [3]; базисом для Model Checking, в первую очередь, служит математический аппарат темпоральной логики (Temporal Logic), то есть такой логики, в которой учитывается фактор времени.

Далее в статье анализируются формальные методы именно в таком – узком смысле. Общей чертой этих методов является то, что они базируются на инвариантоориентированном подходе, другими словами, в их основе лежит выявление и контроль соблюдения инвариантов [4].

1.2. Ограничения и проблемы использования формальных методов

По статистике [5] для многих критических приложений этап тестирования занимает больше двух третей всего времени, затрачиваемого на разработку компьютерной системы. Однако даже после завершения тестирования отсутствует полная уверенность в том, что все дефекты проектирования были выявлены и устранены (или в том, что все требования технического задания были полностью и корректно выполнены). Рассматриваемые в узком смысле формальные методы предлагают подход, при котором корректность программы подтверждается не тестированием, а формальным доказательством с использованием математического аппарата (Event-B) или же автоматической верификации поведения модели системы (Model Checking). По этой причине область их применения ограничивается цифровыми системами управления и управляющими программами, выполняющими относительно простые вычисления, но имеющие большое число состояний и реализующие сложную логику переходов между ними.

Несмотря на то, что основные положения формальных методов были сформулированы более 20 лет назад, а их применение рекомендовано рядом стандартов и нормативных документов [6, 7], они до сих пор не получили широкого практического распространения. На сегодняшний день формальные методы находят применение лишь при разработке ограниченного класса систем (или части их функциональности) с высокими требованиями к надежности (безотказности) и функциональной безопасности. Наиболее частым упоминанием успешного применения формального метода B являются автоматическая система управления линией №14 Парижского метро (1998 г.) и автоматическая система управления экспрессом Парижского аэропорта (2006 г.) [8]. Классическим примером успешного использования метода Model Checking является верификация протокола когерентности кэш-памяти стандарта шины IEEE Futurebus+ (стандарт IEEE 896.1-1991), в результате которой был обнаружен ряд ошибок, не выявленных при традиционном тестировании [3]. Однако эти примеры скорее носят демонстративный характер и показывают потенциальную возможность применения формальных методов для решения инженерных задач разработки сложных компьютерных систем и программного обеспечения.

Одним из факторов, препятствующих использованию формальных методов, являются высокие требования, предъявляемые к опыту и квалификации разработчиков, а также к их специальной математической подготовке. Кроме того, применение формальных методов требует внесение изменений в организацию и процесс выполнения проекта. Другим фактором, ограничивающим их использование, является некоторый консерватизм идеологов программирования и разработчиков систем, состоящий в естественной приверженности к традиционным подходам и технологиям.

Тем не менее, с формальными методами, как со средством, способным превратить «искусство программирования (проектирования)» в «науку программирования (проектирования)», по-прежнему связаны большие ожидания. Об этом свидетельствует и тот факт, что лауреатами премии Тьюринга 2007 года, присуждаемой научно-образовательным сообществом Association for Computing Machinery за значительный вклад в развитие компьютерных наук и технологий, их практическое внедрение, стали Э. Кларк (университет Карнеги-Меллона), А. Эммерсон (Техасский университет в Остине) и Дж. Сифаркис (Гренобльский университет, Франция) за достижения в области Model Checking. Кроме того, в последние годы значительно возросло количество крупных исследовательских проектов в области формальных методов и их практического применения (в частности, RODIN, DEPLOY[1]), которые финансируются международными и европейскими исследовательскими программами и в которые вовлечены многие компании-разработчики программного обеспечения и компьютерных систем (Bosh, Siemens, IBM, Microsoft, Nokia и др.). Сейчас речь идет уже не только о Model Checking, а о model-based (model-driven) development, т.е. более широком использовании формальных методов, основанных на модельной парадигме, для сквозной поддержки (или реализации) процессов разработки.

1.3. Цель и структура статьи

В рамках данной работы авторы пытаются найти ответы на ключевые вопросы, относящиеся к рассматриваемой проблеме: «Насколько «формальны» формальные методы?», «Все ли процедуры формальных методов доведены до уровня инженерных методик?», «Какие факторы препятствуют широкому распространению формальных методов в задачах инжиниринга программного обеспечения и компьютерных систем?». При этом поиск ответов осуществляется на основе анализа особенностей формального метода Event-B и Model Checking подхода.

Таким образом, целью данной статьи является исследование степени формализованности основных процедур этих методов, а также уточнение изменений жизненного цикла программного обеспечения, которые обусловлены их применением. Статья структурирована следующим образом: во втором разделе рассматривается понятие инварианта и связанных с ним элементов таксономии, важных для применения формальных методов (система инвариантов, метрики, синтез инвариантов). В третьем разделе проводится анализ метода формальной разработки Event-B, уточняются модель и процессы жизненного цикла, а также особенности его использования при создании надежных и безопасных систем. Четвертый раздел посвящен рассмотрению аналогичных вопросов применительно к Model Cheking подходу. В заключении обсуждаются результаты и направления дальнейших исследований.

2. Инварианты и их свойства

2.1. Сущность понятия «инвариант». Классификация инвариантов

Ключевым элементом – концептом формальных методов разработки и верификации, таких как Event-B, Model Checking и др., является инвариант. Инвариа́нт – термин, используемый

[1] http://www.event-b.org/.

в математике и физике, а также в программировании, обозначает нечто неизменяемое [9]. Он обобщает используемые в технической диагностике понятия «контрольный признак», «контрольное соотношение», нарушение которого говорит об отказе системы, а выполнение или сохранение в некоторых допустимых пределах подтверждает с определенной вероятностью ее работоспособное состояние.

Простейшими примерами инвариантов здесь могут быть неизменная четность кодов при введении специального бита четности, фиксированное значение сигнатуры для схем встроенного тестирования и др.

Инвариантом в Event-B и Model Checking называется свойство системы (логическое условие, предикат), которое должно всегда выполняться (быть истинным) в процессе функционирования на всем пространстве возможных состояний. Инварианты используются для проверки и подтверждения работоспособного состояния системы путем [2–4]:

– выявления и фиксации ограничений на значения системных переменных (задание диапазона допустимого изменения) (точностные инварианты);

– определения неизменности физической размерности системных переменных при их преобразованиях в процессе вычислений или ограничений на допустимое изменение размерности с учетом известных законов физики (семантические инварианты);

– контроля поведения системы: нахождения в определенном состоянии, выхода из него, переходов между состояниями при выполнении разных функций (функциональные инварианты);

– задания и проверки выполнения необходимых пред- и постусловий для возникновения событий, вызова процедур или функций (логические инварианты).

При использовании формальных методов инварианты могут быть представлены в виде уравнений или неравенств, логических высказываний (предикатов) или арифметико-логических выражений.

2.2. Характеристики (метрики) инвариантов

Задачи формулирования инвариантов и их «извлечения» из требований являются наименее формализованными при использовании формальных методов Event-B и Model Checking. Для оценки качества и корректности как отдельных инвариантов, так и их некоторой совокупности (системы инвариантов [10]), могут быть использованы следующие (свойства) характеристики:

1) ясность (лаконичность);

2) критичность (насколько важным является соблюдение данного инварианта);

3) трассируемость (какому требованию соответствует рассматриваемый инвариант и какое нарушение данного требования он контролирует);

4) формальное представление – вид инварианта (уравнение, неравенство, логическое высказывание, арифметико-логическое выражение, высказывание темпоральной логики и т.п.);

4) назначение (ограничивает диапазон допустимых значений некоторой системной переменной, определяет совместность событий или же задает логику изменения состояний системы), определяемое его типом (точностный, логический, функциональный, семантический);

5) глобальность (должен быть истинным постоянно на всем множестве состояний или же только при нахождении системы в определенном состоянии);

6) подмножество контролируемых требований (полнота контроля, степень покрытия);

7) достоверность контроля требования или требований, возможные ошибки контроля.

Указанные характеристики являются базой для введения метрик – показателей качества инвариантов, которые оценивают степень совершенства (достижимости) соответствующих характеристик. Они задают шкалу и методику измерения этих характеристик.

2.3. Система инвариантов

Инвариант позволяет проверять выполнение одного или нескольких свойств системы и соответствующих им требований. Следует отметить, что все требования, предъявляемые к разрабатываемой системе, как функциональные, так и нефункциональные, потенциально являются инвариантами. Однако не все требования могут быть легко представлены в виде, пригодном для использования в качестве инвариантов в рамках формальных методов. Кроме того, не все требования целесообразно представлять в виде инвариантов. Как следует из [2, 3], инварианты отражают ограничения не на статические свойства системы, а на её поведение во времени (динамические характеристики), которое должно быть ограничено в некоторых допустимых рамках или подчинено определенной логике. Для систем критического применения, при разработке которых активно используются формальные методы, инварианты должны контролировать соблюдение свойств, связанных, прежде всего, с функциями безопасности.

В связи с этим существует проблема определения такой совокупности инвариантов, которая обеспечила бы верификацию проекта и контроль системы с заданной достоверностью.

Система инвариантов представляет собой согласованный набор инвариантов, позволяющих контролировать корректное поведение разрабатываемой системы на всем множестве её возможных состояний. Для анализа качества системы инвариантов могут быть использованы следующие критерии [10]:

1) полнота системы инвариантов;

2) минимальная достаточность для контроля свойств системы;

3) избыточность, например, когда инвариант I1 задан в виде (a ИЛИ b), а инвариант I2 – (b);

4) согласованность (непротиворечивость). Примером противоречивых инвариантов является ситуация, когда инвариант I1 определяет некоторое условие (a), а инвариант I2 – (НЕ a), или же I1 – (a ИЛИ b), а I2 – НЕ a.

В [10] также предложены метрики полноты, минимальности, допустимой и недопустимой избыточности, которые позволяют осуществлять оценку и выбор оптимальной по некоторому критерию системы инвариантов.

2.4. Синтез инвариантов

Следует подчеркнуть, что сама задача генерации (поиска, синтеза) инвариантов не является формальной. Ее решение зависит от опыта и квалификации разработчика, эксперта и носит в некотором смысле характер искусства, что в определенной мере входит в противоречие с самой сутью формальных методов. В то же время при наличии достаточной информации для вычисления метрик отдельных инвариантов оценка качества системы инвариантов может представлять собой достаточно простую и формальную процедуру.

Синтез инвариантов может производиться двумя способами при решении разных задач и на основе разной информации:

1) на основе анализа исходных требований (спецификации) к системе – при ее разработке и пошаговой детализации, начиная от вербальной спецификации и заканчивая генерацией программного кода с доказательством корректности проекта;

2) на основе анализа проекта системы и выявления ее свойств, которые могут быть представлены инвариантами – при верификации законченного проекта.

Первый способ может быть реализован в такой общей последовательности П1:

– нормализуется множество требований $MT = \{T_i\}$ к системе (выбираются и уточняются приоритетные);

– для каждого из требований T_i или их подмножества ΔT определяется «покрывающий» инвариант I_j или «покрывающее» подмножество инвариантов ΔI_r;

– решается задача покрытия требований множеством инвариантов $MI = \{I_j\}$ и определяются минимальные системы инвариантов SI_q, принадлежащие MI;

– выбирается оптимальная по заданному критерию система инвариантов SI_{opt}.

При данной последовательности неясным является вопрос, каким образом (какой архитектурой) обеспечивается функциональность системы на основе системы инвариантов SI_{opt}.

Альтернатива такого подхода – общепринятая для формальных методов процедура пошаговой детализации П2, когда инварианты определяются на каждом шаге. Тогда на каждом шаге возможны постановка и решение локальной задачи получения минимальной подсистемы инвариантов SI_h.

Второй способ реализуется либо в такой общей последовательности, как и П1, либо ее модификации, основанной на формировании множества инвариантов не по требованиям спецификации, а по результатам анализа проекта и выявления неизменных соотношений. К числу таких соотношений (инвариантов) могут относиться, например, семантические инварианты, контролирующие корректность размерности переменных. Данный пример относится к числу универсальных инвариантов, которые могут применяться независимо от конкретного приложения. Оптимизация множества инвариантов может быть выполнена аналогично последним двум операциям процедуры П1.

Далее, с учетом описанных понятий и процедур, проведем анализ методов формальной разработки и верификации в контексте моделей жизненного цикла систем.

3. Метод формальной разработки Event-B

3.1. Сущность и основные принципы

Метод Event-B базируется на использовании нотации абстрактной машины AMN (Abstract Machine Notation) [11, 12] и позволяет получить программный код путем разработки и поэтапной детализации формальной спецификации дискретных систем. Event-B формализует процесс описания свойств и динамического поведения систем, а также обеспечивает контроль за соблюдением этих свойств в процессе функционирования на основе механизма предусловий. В качестве математического аппарата используются логика предикатов, Булева алгебра и теория множеств. При использовании формального метода Event-B спецификация системы представляется в виде её формальной модели (машины), основными элементами которой являются:

– набор системных переменных (variables), конкретное значение которых отражает определенное состояние системы (state);

– контекст (context), представленный в виде набора системных констант;

– инварианты (invariants) – набор свойств (условий), истинность которых должна всегда соблюдаться в процессе функционирования системы;

– события (events), возникающие внутри системы или за её пределами и переводящие систему из одного состояния в другое путем выполнения системой определенных операций, изменяющих значения системных переменных в качестве реакции на каждое конкретное событие;

– набор предусловий (guards) для каждого события, запрещающих его возникновение, если в результате реакции системы на это событие произойдет нарушение инвариантов.

Применение Event-B позволяет повысить качество требований к системе (снизить вероятность дефектов в требованиях), а также значительно сократить или исключить полностью дефекты проектирования, гарантируя корректное поведение системы в рамках используемой системы инвариантов. Ключевыми принципами формального метода Event-B являются следующие.

Рефайнмент (refinement) или детализация, предполагающая поэтапный переход от более абстрактной модели системы к более конкретной. Детализация выполняется путем добавления новых событий или изменения существующих, добавления новых операций, переменных, предусловий, инвариантов или констант.

Автоматическое доказательство, выполняемое путем перебора всех событий и инвариантов, и математического доказательства того, что при возникновении каждого события с учетом механизма предусловий не происходит нарушения ни одного инварианта. При этом на каждом новом этапе детализации выполняется доказательство только новых теорем.

3.2. Жизненный цикл разработки систем при использовании Event-B

Общий подход к разработке формальных спецификаций с использованием нотации Event-B заключается в поэтапной детализации системы с сохранением её основных свойств, которые фиксируются с помощью инвариантов. Разработка формальной спецификации начинается с выделения основных свойств и ограничений (инвариантов) системы, наиболее важных для реализации требуемой функциональности с учетом предметной области. Эти свойства должны быть учтены при разработке абстрактной модели. В соответствии с эмпирическим правилом объем спецификации абстрактной модели не должен превышать двух страниц.

Далее определяется логическая последовательность реализации оставшихся требований к системе с учетом их важности и взаимосвязи, и начинается многоэтапная процедура детализации. Целью проведения очередной детализации является учет очередного требования или группы требований. В качестве аналогии процесса детализации Ж.-Р. Абриаль [8] рассматривает взгляд на систему через микроскоп. При увеличении коэффициента приближения микроскопа проявляются новые детали системы, которые не были видны ранее. Таким образом, каждый шаг детализации – это увеличение коэффициента приближения микроскопа.

При спецификации требований к системе выполняется их классификация. В [13] рекомендуется все требования к системе разделять на функциональные (FUN), требования к аппаратному обеспечению и оборудованию (EQP), требования, важные для безопасности (SAF), требования, связанные с живучестью системы и её работой в условиях деградации (DEG), требования, связанные с временными ограничениями и допустимыми задержками (DEL). Каждому требованию целесообразно присвоить уникальный идентификатор для того, чтобы в дальнейшем, при тестировании системы, иметь возможность выполнить их трассировку.

В рамках формального метода Event-B требования к безопасности, допустимому уровню деградации и к временным характеристикам преобразуются, как правило, в инварианты, т.е. условия, которые должны постоянно выполняться. Оставшиеся требования находят свое отражение при разработке модели системы.

В результате применения формального метода Event-B видоизменяется жизненный цикл разработки системы (рис. 1). Основные трудозатраты приходятся на этап спецификации требований, который плавно переходит в этап проектирования, а затем и кодирования.

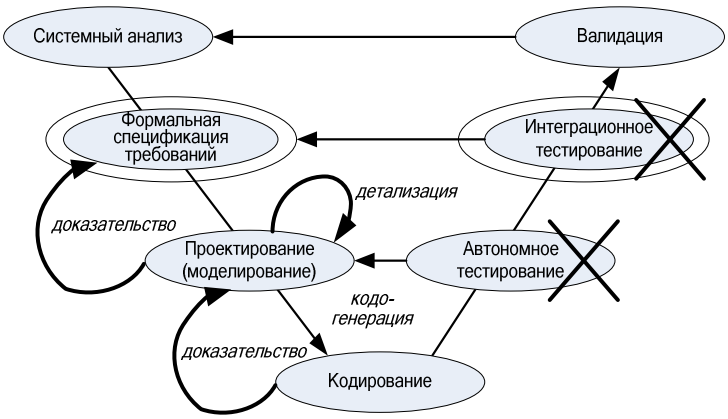

Рис. 1. Изменение модели жизненного цикла при использовании формального метода Event-B

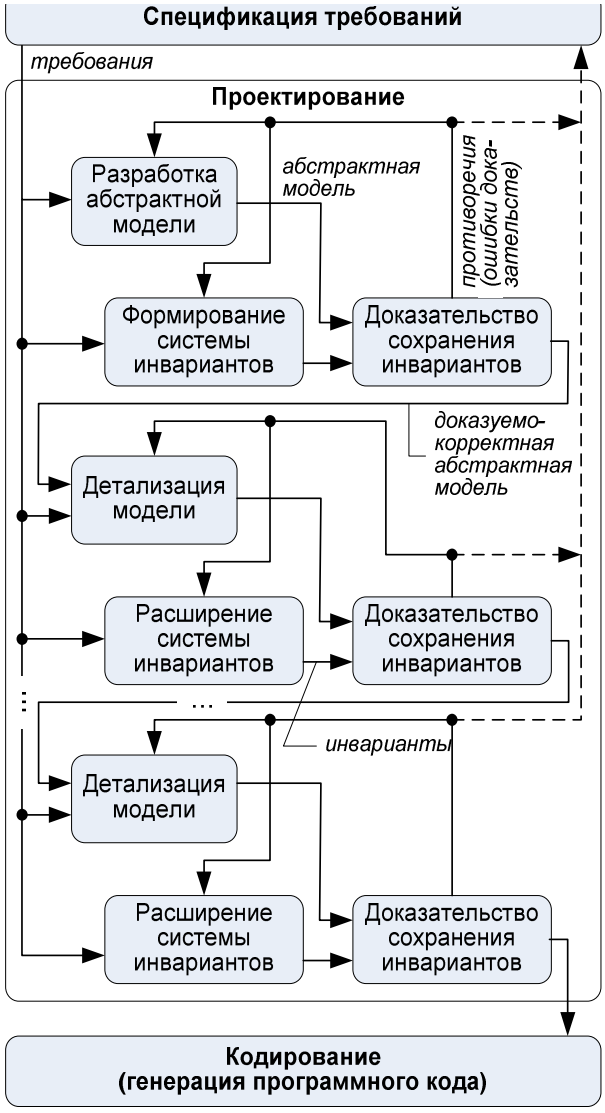

Рис. 2. Этап проектирования модели Event-B

Этап проектирования является итерационным (рис. 2). В рамках каждой итерации выполняются детализация модели системы и доказательство её корректности. При этом отсутствует четкая граница между этапом спецификации требований и этапом проектирования, а потребность в автономном и частично в интеграционном тестировании отпадает.

Переход на этап кодирования в идеальном случае осуществляется путем автоматической генерации (с помощью соответствующих утилит) программного кода из последней детализованной модели системы. Наиболее легко этот переход выполняется для языка программирования Ada, синтаксис которого фактически совпадает с синтаксисом, принятым в *B*-нотации.

Исследователи, исповедующие метод Event-B, настаивают на том, что модель Event-B не является программой, а, следовательно, требует особого инженерного подхода к разработке. Тем не менее нотация Event-B поддерживает многие операторы и конструкции классического программирования (арифметико-логические операции, операторы присваивания, условные операторы, циклы, операторы ветвления и др.), а также реализует концепцию программирования, основанную на событиях [14], так называемую «event-driven programming». Подход к поэтапной детализации разрабатываемой системы, используемый в рамках Event-B, широко применяется и в современной практике программирования. Например, управление программными проектами с использованием технологии SCRUM[2] предполагает поэтапную реализацию требований к программному обеспечению в терминах релизов (release) и спринтов (sprint).

Кроме того, в последнее время значительное внимание уделяется вопросам реализации парадигмы объектно-ориентированного проектирования при разработке моделей Event-B [15, 16].

[2] www.scrumalliance.org/pages/what_is_scrum.

3.3. Анализ особенностей применения Event-B

Целью применения Event-B является минимизация или полное исключение дефектов проектирования в рамках используемой системы инвариантов. В то же время метод плохо подходит для создания систем, устойчивых к физическим отказам элементов [10]. Прежде всего это связано с противоречием, возникающим из-за особенностей самого метода. Event-B предназначен для создания корректных (т.е. «идеальных») систем, в то время как отказоустойчивая система допускает некорректное поведение (вследствие возникновения отказов в системе), которое должно быть парировано.

Применение Event-B фактически ограничено классом дискретных систем управления, алгоритм которых может быть описан с использованием натуральных чисел (хотя известны работы, в которых делаются попытки расширить применение Event-B для чисел с плавающей запятой [17]), элементов дискретной математики и теории множеств. При использовании формального метода Event-B существует опасность того, что «бесконечный» процесс тестирования, наблюдаемый при традиционном подходе к разработке ПО, будет заменен «бесконечным» процессом детализации. Действительно, на сегодня отсутствуют методики, формализующие процесс детализации модели Event-B.

Открытыми остаются вопросы: «Сколько шагов детализации необходимо для построения адекватной модели системы?», «Какой должна быть абстрактная модель системы?», «Чем и насколько должна отличаться более конкретная модель от более абстрактной?», «Каким образом (в каком виде) и в какой последовательности должны быть учтены требования к системе при выполнении детализации?».

Для ответа на эти вопросы может потребоваться проведение ряда практических экспериментов по анализу диверсности проектов Event-B. Вполне очевидно, что разные команды разработчиков, имея одну и ту же исходную спецификацию требований, получат модели Event-B, отличающиеся друг от друга и по содержанию, и по количеству выполненных шагов детализации. Являются ли такие модели тождественными?

С одной стороны, если метод Event-B гарантирует соблюдение инвариантов и правильность детализации, то не все ли равно, каким образом, начиная с какой абстрактной модели и за какое количество шагов детализации была получена финальная модель Event-B. С другой стороны, диверсные модели Event-B, скорее всего, будут использовать и разные системы инвариантов. А раз так, то разработанные модели будут корректными только в рамках используемой системы инвариантов, а не «глобально» корректными.

Как следует из приведенных выше рассуждений, особое внимание при использовании формального метода Event-B заслуживает система инвариантов, которая является критическим элементом модели Event-B. К сожалению, нотация Event-B не описывает методики формирования инвариантов (извлечения инвариантов из требований), оценки качества (корректности, достаточности, полноты и т.п.) инвариантов [10], а также их верификации. Принципиальными вопросами здесь являются: «Все ли требования являются инвариантами?», «Если нет, то какие требования должны быть зафиксированы с помощью инвариантов?», «Каким образом возможно доказательство корректности отдельного инварианта и полноты всей системы инвариантов?».

Одним из ключевых вопросов практического применения Event-B является также инструментальная поддержка, включая утилиты и средства автоматического доказательства теорем (так называемый automatic theorem prover), их качество и также верифицируемость. Кроме того, на практике не все теоремы могут быть доказаны автоматически.

4. Метод формальной верификации Model Checking

4.1. Сущность и основные принципы

Model Checking – метод автоматической формальной верификации систем с конечным числом состояний, который позволяет проверить, удовлетворяет ли заданная модель сис-

темы формальной спецификации, определяющей требуемое поведение [3]. Формальная спецификация поведения системы представляется в виде высказываний темпоральной логики. Темпоральная логика (ТЛ) – это логика, учитывающая причинно-следственные связи в условиях времени. ТЛ используется для описания последовательностей событий и их взаимосвязи на временной шкале и позволяет описать требуемое поведение системы во времени. В качестве примера формальной спецификации, проверяемой с помощью Model Checking, могут быть использованы высказывания: «Система никогда не попадет в состояние S2, минуя состояние S1», «Система когда-нибудь перейдет в состояние S3», «В системе наступит момент, когда значение переменной V станет равным нулю» и т.п. Такие высказывания, по сути, являются аналогом инвариантов формального метода Event-B. В качестве модели системы в Model Checking используется модель Крипке [3], которая описывает множество состояний, множество переходов между состояниями, а также задает функцию, помечающую каждое состояние набором свойств, истинных в этом состоянии. Пути в модели Крипке соответствуют вычислениям системы.

Model Checking применяется, как правило, для верификации аппаратного обеспечения, параллельных систем, коммуникационных протоколов, а также ограниченного класса программного обеспечения, представленного, в основном, автоматными программами, т.е. программами, поведение которых может быть полностью специфицировано с помощью конечных автоматов, например, автомата Мили. Алгоритмы верификации Model Checking базируются на полном обходе пространства состояний модели. Для каждого состояния проверяется, удовлетворяет ли оно предъявляемым требованиям. Если результат верификации негативный, то пользователю предоставляется трасса, содержащая ошибку. Она строится в качестве контрпримера для нарушаемого свойства.

4.2. Жизненный цикл разработки систем при использовании Model Checking

При использовании метода Model Checking вместо тестирования системы выполняется формальная верификация её модели. Кроме того, добавляются два дополнительных процесса:

1) моделирование, которое заключается в приведении проектируемой системы к такому формальному виду, который был бы приемлем для инструментальных средств верификации моделей программ, например, SPIN или SMV [18]. При разработке сложных программных комплексов получить модель Крипке с конечным числом состояний напрямую из программного кода практически невозможно. В этом случае требуется абстракция системы;

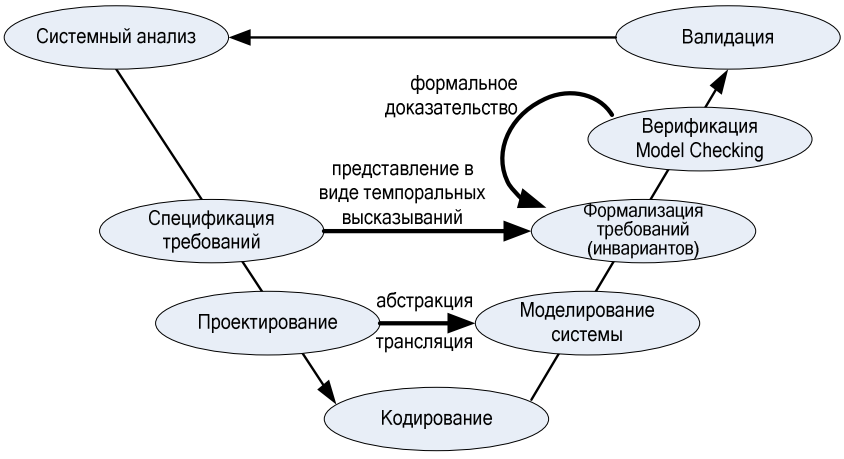

Рис. 3. Изменение модели жизненного цикла при использовании формального метода Model Checking

2) формализация требований, в рамках которой выполняется представление исходных требований к системе в виде высказываний темпоральной логики, например, с использованием логики линейного времени LTL (Linear Temporal Logic) или логики деревьев вычислений CTL (Computational Tree Logic).

Вариант модели жизненного цикла при использовании Model Checking представлен на рис. 3. Для формализации процесса моделирования традиционным является подход, в

рамках которого выполняется трансляция исходного программного кода, написанного на существующих языках программирования (Ada, C/C++, Java и др.), в формат (например, в программу на языке моделирования PROMELA, который применяется для описания модели Крипке), используемый средствами автоматической формальной верификации. Для этих целей разрабатываются специальные инструментальные средства – трансляторы.

Возможности таких трансляторов постепенно расширяются, о чем свидетельствует анализ работ [19–21]. Однако эти трансляторы до сих пор далеки от совершенства, требуют от программистов соблюдения определенных соглашений и использования ограниченных приемов программирования (использование ограниченного набора операторов и структур данных, добавление специальных комментариев (аннотаций) в текст программы и т.п.) и не могут быть успешно применены для всего многообразия разрабатываемых программных средств. Очевидный недостаток такого подхода заключается в том, что проверка корректности программ выполняется постфактум, т.е. Model Checking не предотвращает дефекты проектирования, а только позволяет их обнаруживать.

Второй подход является прямо противоположным и базируется на создании программ (моделей программ) непосредственно с использованием языка моделирования PROMELA или других высокоуровневых языков моделирования (например, SPL [22]), пригодных для непосредственной проверки утилитами формальной верификации. После успешной верификации высокоуровневая модель программы транслируется в один из традиционных языков программирования. Такой подход фактически соответствует модели жизненного цикла при использовании формального метода Event-B (рис. 1).

Однако используемые языки моделирования не обладают всеми возможностями и выразительностью современных языков программирования и, следовательно, могут быть использованы лишь для ограниченного класса управляющих программ или отдельных программных модулей.

4.3. Анализ особенностей применения Model Checking

Несмотря на хорошие теоретические свойства методов верификации моделей программ и их практическую ценность, применение Model Checking сопряжено с рядом трудностей.

При использовании Model Checking проверяется не сама система, а её модель. Основная трудность моделирования – выбрать достаточный уровень абстракции и сохранить при этом важные детали моделируемой системы, т.е. существуют риски «второго пришествия» проблемы размерности, решение которой явилось одним из побудительных мотивов разработки формальных методов. Классический подход к использованию Model Checking направлен на обнаружение ошибок и дефектов проектирования, но не на их предупреждение или недопущение. Кроме того, при переходе от программной (или аппаратной) реализации системы к её модели возможно внесение дополнительных дефектов.

Метод проверки на модели дает возможность убедиться, что модель проектируемой системы соответствует заданной спецификации, однако определить, охватывает ли заданная спецификация все свойства (инварианты), которыми должна обладать система, невозможно. Трудность задания проверяемых свойств заключается в необходимости их правильного и исчерпывающего выражения с использованием операторов темпоральной логики. Здесь так же, как и для метода Event-B, имеют место проблемы, связанные с обеспечением и верификацией полноты, корректности, достаточности системы инвариантов.

Несмотря на то, что в литературе описан пример успешного использования Model Checking для верификации системы с числом состояний около 10^{120} [5], одна из проблем применения метода связана с эффектом «комбинаторного взрыва» в пространстве состояний. Этот эффект возникает в системах, состоящих из многих компонентов, взаимодействующих друг с другом, а также в системах, использующих структуры данных, способные принимать большое число значений (многие программы и сетевые протоколы задаются

для бесконечного количества различных конфигураций, используют переменные над бесконечными типами данных).

Широкое практическое использование Model Checking возможно лишь при условии, что будут разработаны более совершенные трансляторы с языков программирования в языки моделирования Model Checking или же утилиты автоматической верификации смогут работать непосредственно с исходным программным кодом.

5. Заключение

5.1. Обсуждение результатов

Необходимо признать, что на сегодняшний день существуют объективные причины как технического, так и технологического характера, препятствующие широкому практическому использованию формальных методов. Применение формальных методов требует привлечение в проект более квалифицированных разработчиков с серьезной математической подготовкой и практическими навыками, позволяющими выразить требования к системе и логику её работы с использованием математического аппарата. По этой причине предельная ситуация, когда классические программисты останутся без работы вследствие масштабного применения формальных методов и соответствующих кодогенераторов, пока еще нереальна.

Системы, разрабатываемые с использованием формальных методов, имеют более «жесткий» жизненный цикл (ЖЦ) и требуют строгого контроля за его соблюдением, а также доказательного перехода от одного этапа ЖЦ к другому. Все еще остается не решенной проблема «последнего шага», т.е. автоматического перехода от модели системы к её программной реализации и наоборот.

В исходном виде существующие формальные методы не поддерживают современные парадигмы программирования и проектирования компьютерных систем и программного обеспечения (объектно-, компонентно-, аспектно-ориентированное проектирование и др.). Наименее формализованными процессами формальных методов являются детализация (refinement) модели системы, формулирование инвариантов и формирование системы инвариантов. Применение методов, аналогичных Event-B, порождает опасность перехода от проблемы «бесконечного тестирования» к проблеме «бесконечной детализации» модели разрабатываемой системы. Другими проблемами данного класса формальных методов являются: неформализованный процесс детализации модели, ограниченная поддержка сложных структур данных и операций с числами, не входящими в множество натуральных чисел, отсутствие доказательства полноты используемой системы инвариантов, необходимость подбора вариантов, сложность построения и проблема верификации автоматических средств выполнения доказательства.

Формальные методы верификации Model Checking также имеют свои недостатки, связанные с ограниченным числом состояний и переходов анализируемой модели, отсутствием универсальных средств автоматической генерации модели Крипке системы из исходного кода, невозможностью доказательства полноты верифицируемых свойств. Сложность построения модели анализируемой системы в ряде случаев может оказаться сопоставимой по сложности с разработкой самой системы. Существенное же упрощение модели приводит к снижению степени её адекватности.

Необходимо также отметить, что совместное применение методов Event-B и Model Checking в рамках одного проекта при использовании одной и той же системы инвариантов является избыточным и нецелесообразным.

5.2. Следующие шаги

Следует отметить, что на сегодняшний день применение формальных методов для всего класса систем и задач нецелесообразно или же невозможно. Тем не менее, возродившийся

в последние годы интерес к формальным методам разработки и автоматического доказательства корректности программного обеспечения позволяет надеяться, что в скором времени появятся новые технологии программирования, сочетающие как простоту и выразительность классических методологий программирования и проектирования, так и строгость формальных методов.

Наибольшее внимание и усилия, на наш взгляд, целесообразно направить на развитие инженерно-методической и технологической поддержки формальных методов. Прежде всего, разработчики нуждаются в инженерных методиках, регламентирующих основные этапы применения формальных методов и их особенности и дающих четкие рекомендации по выполнению наименее формализованных процедур детализации моделей, спецификации инвариантов, обеспечению и оценке качества и полноты системы инвариантов. Критическим является разработка автоматических средств трансляции и кодогенерации, предназначенных для перехода от модели системы к её программной реализации, и, наоборот, для представления программного кода в виде модели, пригодной для формальной верификации.

Кроме того, необходимо создание интегрированных сред формальной разработки и верификации нового поколения (например, платформа Rodin [23]), дружественных для программистов и обеспечивающих интеграцию формальных методов с традиционными методологиями проектирования, а также скрывающих математическую сложность формальных методов за интуитивно-понятным интерфейсом разработки наподобие UML.

Одной из ключевых задач является уменьшение доли «искусства» при формировании системы инвариантов. Вопрос заключается не в том – наука это или искусство, а в том, сколько места остается искусству. Интересным, с учетом этого, является вопрос о методах нечеткой математики [24]: это альтернатива или дополнение к формальным методам? Учитывая генетическую близость теории инвариантов и классической диагностики, а также то, что при синтезе и реализации инвариантов возможны ошибки и неточности, проблему «Кто будет сторожить сторожей?» можно сформулировать как проблему «Кто будет разрабатывать, проверять и применять инварианты для инвариантов?».

Требуют своего уточнения границы формальных методов. Нет споров о том, что аппарат теории марковских процессов – это формальный (в широком смысле) метод. Однако и для него существует «незаполненное (неформализованное) пространство» при детализации состояний системы, которое может быть лишь частично формализовано [25]. В этой связи можно сказать о необходимости разработки «главного» формального метода, который позволит корректно декомпозировать любую задачу на подзадачи, где работают без всяких оговорок разные формальные методы.

СПИСОК ЛИТЕРАТУРЫ

1. Schneider S. Palgrave. The B-Method: An Introduction / Schneider S. – Hampshire: Cornerstones of Computing series, 2001. – 537 p.
2. Abrial J.-R. Modeling in Event-B: System and Software Engineering / J.-R. Abrial. – Cambridge: Cambridge University Press, 2009. – 612 p.
3. Кларк Э.М. Верификация моделей программ: Model Checking / Кларк Э.М., Грамберг О., Пелед Д.; пер. с англ.; под ред. Р. Смелянского. – М.: МЦНМО, 2002. – 416 с.
4. Инвариантно-ориентированная оценка качества программного обеспечения космических систем / [Конорев Б.М., Манжос Ю.С., Харченко В.С. и др.]; под ред. Б.М. Конорева, В.С. Харченко. – Харьков: ХАИ, Госцентркачества, 2009. – 224 с.
5. Фатрелл Р.Т. Управление программными проектами. Достижение оптимального качества при минимуме затрат / Фатрелл Р.Т., Шафер Д.Ф., Шафер Л.И. – М: Вильямс, 2004. – 1136 с.
6. Bowen J.P. Formal Methods in Safety-Critical Standards / J.P. Bowen // Proc. Software Engineering Standards Symposium (SESS'93) // IEEE Computer Society Press. – Brighton, UK, 1993. – P. 168 – 177.

7. Bowen J.P. Safety-Critical Systems, Formal Methods and Standards / J.P. Bowen, V. Stavridou // Software Engineering Journal. – 1993. – Vol. 8(4). – P. 189 – 209.
8. Lecomte T. Formal Methods in Safety-Critical Railway Systems [Електронний ресурс] / T. Lecomte, T. Servat, G. Pouzancre // Proc. 10th Brasilian Symposium on Formal Methods, (Ouro Preto (Brazil), 29–31 August 2007). – Режим доступу: http://deploy-eprints.ecs.soton.ac.uk/8/1/fm_sc_rs_v2.pdf.
9. Визгин В.П. Развитие взаимосвязи принципов инвариантности с законами сохранения в классической физике / Визгин В.П. – М.: Наука, 1972. – 240 с.
10. Abrial J.-R. The B-Book: Assigning Programs to Meanings / Abrial J.-R. – Cambridge: Cambridge University Press, 1996. – 853 p.
11. Diller A. Z: An Introduction to Formal Methods / Diller A. – Wiley, 1994. – 354 p.
12. Abrial J.-R. Formal Method Course / Abrial J.-R. – Zürich, 2005. – 219 p.
13. Samek M. Practical UML Statecharts in C/C++: Event-Driven Programming for Embedded Systems / Samek M. – Newnes, 2008. – 728 p.
14. Snook C. UML-B and Event-B: an integration of languages and tools [Електронний ресурс] / C. Snook, M. Butler // The IASTED International Conference on Software Engineering (SE'2008). – 2008. – Режим доступу: http://eprints.ecs.soton.ac.uk/14926/4/SnBu08.pdf.
15. Edmunds A. Linking Event-B and Concurrent Object-Oriented Programs / A. Edmunds, M. Butler // Electronic Notes in Theoretical Computer Science (ENTCS). – 2008. – Vol. 214. – P. 159 – 182.
16. Tarasyuk O. Principles of Formal Methods Integration for Development Fault-Tolerant Systems: Event-B and FME(C)A / O. Tarasyuk, A. Gorbenko, V. Kharchenko // MASAUM Journal of Computing. – 2009. – Vol. 1, Issue 3. – P. 423 – 429.
17. Barrett G. Formal methods applied to a floating-point number system / G. Barrett // IEEE Trans. Software Eng. – 1989. – Vol. 15, N 5. – P. 611 – 621.
18. Dwyer M. Translating Ada Programs for Model Checking: A Tutorial / M. Dwyer, C. Pasareanu, J. Corbett // Technical report KSU CIS TR-98-12. – Manhattan (USA): Kansas State University, 1998. – 17 p.
19. Visser W. Tutorial on Software Model-Checking [Електронний ресурс] / W. Visser // Proc. 17th IEEE International Conference on Automated Software Engineering (ASE'2002). – 2002. – Режим доступу: www.visserhome.com/willem/presentations/ASE2002TutSoftwareMC-fonts.ppt
20. Model Checking Programs [Електронний ресурс] / W. Visser, K. Havelund, G. Brat [et al.] // Automated Software Engineering Journal. – 2003. – Vol. 10, N 2. – Режим доступу: http://ti.arc.nasa.gov/people/wvisser/ase00FinalJournal.pdf.
21. Formal Methods: Practice and Experience / J. Woodcock, P.-G. Larsen, J. Bicarregui [et al.] // ACM Computing Surveys. – 2009. – Vol. 41, N 4. – P. 1 – 36.
22. Madhavapeddy A. SPLAT: A Tool for Model-Checking and Dynamically-Enforcing Abstractions / A. Madhavapeddy, D. Scott, R. Sharp // Proc. 12th International SPIN Workshop on Model Checking of Software, LNCS 3639. – Berlin: Springer-Verlag, 2005. – P. 277 – 281.
23. Abrial J.-R. A System Development Process with Event-B and the Rodin Platform / J.-R. Abrial // LNCS 4789. Formal Methods and Software Engineering. – 2007.
24. Раскин Л.Г. Нечеткая математика. Основы теории. Приложения / Л.Г. Раскин, О.В. Серая. – Харьков: Парус, 2008. – 352 с.
25. Тарасюк О.М. Многошаговая процедура построения графа состояний при исследовании вычислительных систем с использованием аппарата марковских процессов / О.М. Тарасюк, Л.С. Усов, В.С. Харченко // Радіоелектронні і комп'ютерні системи. – 2005. – № 4 (12). – С. 75 – 81.

Стаття надійшла до редакції 20.11.2009