Технология проектирования и верификации распределенных встроенных бортовых систем

I.V. Shoshmina

1. Актуальность

Сложность промышленного программного обеспечения постоянно растет. Современный программный продукт может содержать десятки миллионов строк кода. Мысленно охватить функционирование таких систем не в состоянии ни один человек даже при использовании современных технологий и методов абстрагирования и управления сложностью. Как следствие, вместе со сложностью растет и число потенциальных ошибок в программах. Только в США финансовые потери от ошибок в программах оцениваются в десятки миллиардов долларов в год. Например, в 2002 г. эти потери оценены в $59,6 млрд. [1].

Особенно трудно поддаются анализу ошибки в параллельных, распределенных, многопоточных программах, характерных для бортовых систем управления техникой. Хорошо известно, что даже в тех случаях, когда функционирование каждой из параллельных взаимодействующих компонент системы абсолютно ясно, человеку трудно понять работу всей параллельной системы, процессы в которой взаимозависимы. При разработке параллельной программы необходимо контролировать возможные комбинации частично упорядоченных событий, что значительно сложнее, чем контроль полностью упорядоченных событий в обычных программах. Параллельные системы, которые работают правильно “почти всегда”, годами могут сохранять тонкие ошибки, проявляющиеся в редких, исключительных ситуациях. Такие ошибки, как правило, невозможно обнаружить тестированием.

В наше время, когда все чаще человеческая жизнь зависит от функционирования автоматических систем, проблема гарантии правильности работы программных и аппаратных компонентов этих систем приобретает первостепенное значение. Надежность и предсказуемость поведения таких систем являются более важными свойствами, чем производительность, модифицируемость, переносимость и т.п. Верификация ПО, как один из основных методов повышения качества программных систем, становится важнейшей областью информатики.

2. Цель работы

Целью данной работы является изложение результатов исследования, направленного на включение формальной верификации в процесс разработки встроенных распределенных программных систем для повышения их надежности. Результатом исследования является методика проектирования программных средств на основе моделей (Model-Driven Engineering) с использованием формальных методов верификации как одного из этапов проектирования.

В последние годы разработан новый подход к верификации – “проверка модели” (model checking), который является очень перспективным для использования в промышленной разработке ПО. “Проверка модели” – это формальная проверка того, является ли данная формула (как правило, формула темпоральной логики) истинной на данной структуре. Структура представляет модель разрабатываемой программной системы. Формула описывает желаемые требования к поведению этой системы. Уже сегодня этот метод используется для проверки сложных объектов, как программного обеспечения, так и аппаратуры. Он позволяет существенно повысить степень уверенности разработчиков в правильности функционирования интегральных схем, протоколов коммуникации, драйверов устройств, бортовых систем управления для автомобилей, самолетов, космических аппаратов. В работе рассматривается методика разработки встроенного ПО именно на основе проверки модели.

3. Процесс проектирования под управлением формальных моделей с поддержкой верификации

Формальная верификация программ – это приемы и методы формального доказательства (или опровержения) того, что модель программной системы удовлетворяет заданной формальной спецификации. Для того чтобы доказать формально какое-либо утверждение относительно работы реальной системы, анализируемая система (реализация) должна быть представлена формальной моделью. Формальная модель обычно проще самой проверяемой системы, это абстракция, в которой отражены наиболее существенные характеристики системы.

Для верификации встроенной системы управления также необходимо построить ее модель. Подобные системы обычно реализуются на языках высокого уровня с формально определенным синтаксисом и кажутся полностью формализованными. Однако с точки зрения семантики это не так. Использование указателей, обработка вещественных чисел с ограниченной точностью, сложные структуры данных, динамическое порождение неограниченного числа потоков и их

взаимодействие – все это приводит к тому, что из текста программы не следует непосредственно полное формальное описание ее поведения.

Поскольку в общем случае процесс построения формальной модели системы не формализуем и не однозначен, имеет смысл выделить класс подсистем, для которых можно автоматически построить адекватную формальную модель. К таким системам, например, относятся реагирующие системы: системы, дающие отклик на внешнее событие в зависимости от своего состояния. Реагирующие системы часто являются частью большей программной системы. Операционные системы, протоколы коммуникации, планировщики, контроллеры, параллельные взаимодействующие программы, системы логического управления, драйверы – все это примеры реагирующих систем.

Консорциумом OMG предложен подход к проектированию программ на основе моделей, или под управлением моделей, названный Model-Driven Engineering (MDE). В MDE формальные модели программных систем первичны в процессе проектирования [2]. Спецификация системы отделена от технологии реализации или платформы. Описание системы существует независимо от конкретики реализации и может быть формально транслировано на большое число платформ (в т.ч., Java, XML, SOAP). Верификация требуемых свойств по методу проверки модели проводится именно по формальной модели программы. Для верификации можно использовать любой из известных верификаторов, реализующий метод проверки на модели. Именно эта стратегия разработки MDE принята в разрабатываемой методологии проектирования встроенных реагирующих систем.

4. Детали практической методологии проектирования ПО на основе формальных моделей

Поскольку проект сложной распределенной программной системы включает в себя множество подсистем, не требующих верификации, либо не влияющих на верифицируемые свойства, будем строить и верифицировать лишь модель ядра системы, отвечающего за управление. Ядро управления представляет собой реагирующую систему. В соответствии с MDE будем строить формальную модель ядра управления на платформенно-независимом языке моделирования UML.

Поведение каждого независимого объекта ядра управления в разработанной методике описывается визуальным формализмом, картами состояний (statechart). Они представлены в языке моделирования UML диаграммами конечных автоматов (UML statemachine diagram) и диаграммами активностей (UML stateactivity diagram). Поведение всех объектов ядра управления является

детерминированным: на каждое уникальное событие карта состояния, представляющая поведение объекта, имеет единственный переход.

Взаимодействие между объектами ядра может происходить разным образом: через каналы (синхронные или асинхронные) или через разделяемые переменные. Из множества диаграмм, предлагаемых в рамках UML, трудно выделить какую-либо из них, отвечающую за описание механизмов взаимодействия. Некоторые авторы предлагают описывать механизмы взаимодействия зависимостями на диаграмме классов (UML class diagram) или диаграмме развертывания (UML deployment diagram) [3].

Верификация проводится для замкнутых систем, т.е. для верификации необходимо моделировать поведение не только ядра управления, но и среды, выдающей события. Поведение среды можно ограничить, задав сценарий, т.е. конечную фиксированную последовательность событий (например, с помощью диаграмм взаимодействия UML). Модель среды в виде сценария может описывать поведение оператора за пультом в соответствии с вариантом использования. Однако такой подход не предусматривает случайного, не предусмотренного стандартными сценариями события, например, нажатия кнопки включения системы пожарной безопасности на подводной лодке, а потому не позволяет выявить тех поведений ядра управления, которые никогда не должны происходить. В разработанной методике моделирование среды выполняется недетерминированной картой поведения, что покрывает все возможные сценарии ее поведения.

Полученная модель системы на языке UML транслируется в язык Promela средства комплексной верификации Spin [4]. Программное средство комплексной верификации Spin, разработанное в исследовательском центре Bell Labs, широко применяется в промышленности для формального анализа свойств разрабатываемых протоколов и бортовых систем. Пакет Spin позволяет строить модели параллельных программ и широкого класса дискретных систем, выразить требуемые свойства их поведения на языке линейной темпоральной логики LTL, автоматически проверить выполнение темпоральных свойств систем на их моделях.

Для спецификации требований к системе разработанная методика предлагает подход, основанный на шаблонах [5]. Использование шаблонов требований скрывает от разработчика формализм, описывающий задаваемое требование. Каждому шаблону требований может быть поставлена в соответствие формула линейной темпоральной логики (LTL), которая системой верификации Spin

переводится в код на языке Promela, соответствующий отрицанию требования.

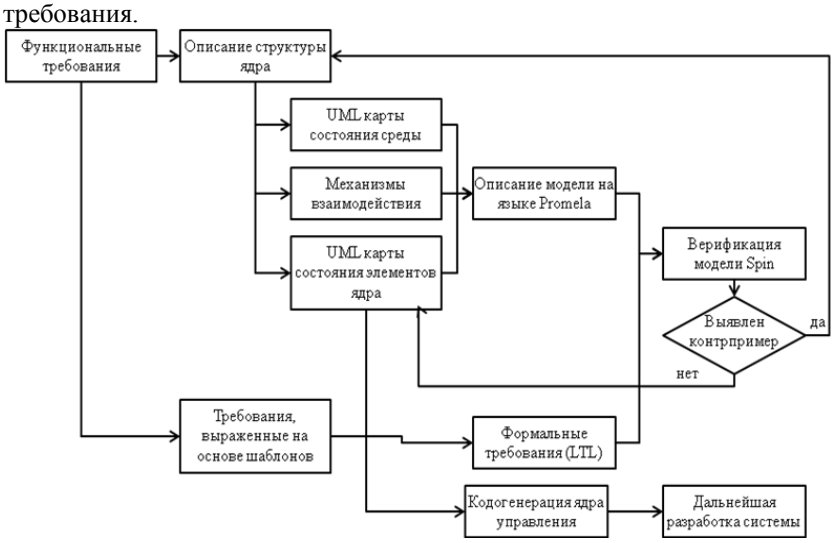

Рис.1 Общая схема процесса проектирования под управлением моделей с поддержкой верификации.

Синхронная композиция модели ядра системы и отрицания требования на Promela верифицируется системой Spin при помощи метода проверки на модели. Если в результате будет обнаружен контрпример, то он может отображаться в виде диаграммы последовательностей (UML sequence diagram). Найденное системой верификации опровержение требования к поведению системы свидетельствует о некорректности построенной модели относительно заданного свойства и является источником информации об ошибке в системе, что облегчает модификацию системы с целью устранения этой ошибки. После модификации модели этапы верификации повторяются вновь. Если ошибок не обнаружено, то модель ядра управления корректна по отношению к данному свойству (рис. 1).

При помощи верификации невозможно гарантировать, что система корректна относительно всех возможных требований, потому что пространство требований – открытое множество. Однако проверка набора всех требований к ядру управления формирует в результате модель, которая имеет только те поведения, которые удовлетворяют специфицированным требованиям корректности поведения системы при недетерминированном поведении среды.

5. Верификация модели системы управления энергоснабжением судна

Данная методика была применена к верификации системы управления энергоснабжением судна. Изначально было принято решение о построении модели на языке Promela по имеющемуся коду. Как уже было сказано ранее, использование языка Promela предполагает абстрагирование от деталей реализации. Построение модели на Promela производилось вручную по следующим причинам:

- сложность автоматического отсечения лишних деталей реализации, которые не требуют моделирования,
- отсутствие структуры программного кода (с точки зрения применения методов верификации: функции управления смешаны с функциями среды, функциями архивирования и т.п.),
- в Promela моделируется замкнутая система — т.е. задается поведение КСУ вместе с поведением среды (датчиков, кнопок и т.п.), поскольку поведение среды в реализации не описывается,
- отсутствие автоматических средств обратного проектирования для Promela с языка C++.

Поведение каждой из подсистем системы управления электроснабжения судна, система опроса и установки датчиков моделировались отдельным процессом. Разделяемая память моделировалась, как массив общих переменных. Был выделен конечный набор сообщений. Обмен сообщениями между процессами производился при помощи асинхронных каналов конечной глубины. В системе опроса и установки датчиков моделировалось поведение физической среды при помощи операторов недетерминированного выбора, т.о. модель описывала любое возможное поведение среды.

Кроме этого была решена задача моделирования справедливого доступа к разделяемым переменным. В реализации справедливый доступ обеспечивается использованием CORBA. Естественно, что сохранение всех деталей реализации привело бы к разрастанию размеров модели. Поэтому было выполнено абстрагирование, при котором были сохранены все свойства обращения к разделяемым переменным. Решение представляет собой справедливую реализацию алгоритма читателей-писателей классической задачи взаимного исключения.

6. Типы проверяемых требований

Для анализируемого класса систем управления была разработана методика выделения требований к поведению таких систем, выполнение которых на модели свидетельствует о достаточно высоком

качестве этой системы. Были выделены четыре типа (множества) требований корректности.

1. Требования корректности, связанные с отсутствием типичных ошибок параллельных процессов (блокировки (зависания) процессов, нарушение свойств безопасности и живости, одновременный доступ к критическим интервалам или доступ к ним не в требуемой последовательности и некоторые другие требования).
2. Требования корректности, вытекающие из технического описания функционирования системы (требования к поведению системы во всех базовых сценариях и режимах функционирования).
3. Требования управляемости системы (достижимость всех состояний, характеризующих выполнение требуемых функций)
4. Требования надежности (невозможность перехода в критические и опасные режимы кроме строго предписанных регламентированных процедур).

7. Ошибки проектирования, найденные в процессе верификации

При верификации построенной модели системы был найден ряд некорректностей. Некоторые из них перечислены ниже.

Одной из часто встречающихся некорректностей распределенных систем является взаимная блокировка процессов. В построенной модели были найдены блокировки процессов, которые возникают из-за переполнения каналов обмена. Поскольку размер модели не позволяет проверить базовые свойства на объемах канала, использующихся на практике, то можно отнести данную ошибку к ограничениям, возникающим при моделировании. Однако выявление данной ошибки позволило обнаружить большое количество избыточных, с логической точки зрения, взаимодействий между процессами, имевшихся в исходной реализации. Данную проблему удалось обойти, удалив лишние пересылки и снизив размерность системы.

В спецификации КСУ утверждается, что выход из режима защиты ГА возможен только при нажатии кнопки «*Сброс защиты*». Для проверки этого требования на языке LTL было сформулировано свойство: G!(q ^ !p), где p — утверждение о том, что кнопка “сброс защиты” нажата, а q — утверждение о том, что система вышла из режима защиты; т.е. никогда не будет так, что система вышла из режима защиты, а кнопка сброса не была нажата. При проверке этого свойства система верификации нашла трассу выхода ГА из режима защиты без нажатия кнопки сброса защиты. Эта некорректность объясняется

отсутствием необходимых переходных режимов и была решена добавлением переходного режима.

8. Выводы

В результате проведенного исследования удалось продемонстрировать необходимость верификации распределенных систем, выполняющих критические функции, в частности, бортовых систем управления. Исследование подтвердило работоспособность разработанной методики проектирования класса встроенных систем управления. В рамках выработанной методики построена модель действующей системы управления энергоснабжением судна и проведена ее верификация, которая позволила обнаружить несколько тонких ошибок, оставшихся в бортовой системе управления после выполнения всех этапов традиционной технологии разработки ПО этого класса, включающей тщательное тестирование сценариев использования будущей системы. Выявлен ряд узких мест, которые требуют дальнейшего исследования, в частности, необходимость автоматизации процесса снижения размера модели, т.е. построения абстракций, достаточных для верификации конкретных свойств. Это вызвано тем, что верификация модели КСУ была серьезно осложнена ее размером. Для всех проверенных свойств приходилось вручную снижать размер модели, абстрагируясь от малосущественных деталей реализации. Для корректного автоматического выполнения этой функции необходимо иметь проект (структуру) программной реализации системы. Наличие кода на языке Promela недостаточно, т.к. ввиду скудности операторов многие разные конструкции исходной реализации на языке Promela имеют одинаковый вид, однако их свертка при построении абстракции различна.

Литература

1. Federal Study, US Dept of Commerce, May, 2002

2. OMG, Architecture Board ORMSC «Model Driven Architecture (MDA)», ormsc/2001-07-01, July 9, 2001

3. Фаулер М., Скотт К. «UML. Основы» СПб: Символ-Плюс, 2002. 192 с.

4. Holzmann G. «Spin Model Checker, The: Primer and Reference Manual», Addison Wesley, 2003

5. Dwyer M.B., Avrunin G. S., Corbett J. C. «Patterns in Property Specifications for Finite-State Verification», The 21st International Conference on Software Engineering ICSE, 1999, стр. 411 – 420