Модел. и анализ информ. систем. Т. **17**, № **4** (2010) 125–136

УДК 519.68/.69

Проектирование программных бортовых систем управления с поддержкой верификации

Шошмина И.В.

Санкт-Петербургский государственный политехнический университет

e-mail: ishoshmina@dcn.ftk.spbstu.ru

получена 25 октября 2010

Ключевые слова: проектирование на основе модели, проверка модели, UML, LTL, Promela

Рассматривается проблема включения метода формальной верификации — проверки модели — в процесс проектирования сложных распределенных программных систем. В качестве центрального объекта проектирования предлагается использовать платформенно-независимую модель, которая строится на основе ядра системы, отвечающего за логическое управление всей системой как единым целым. Подход позволяет повысить качество разрабатываемого программного обеспечения и гарантировать соответствие задаваемой спецификации. Предлагаемая методика опробована на реальной системе управления энергоснабжением судна.

1. Введение

В современном мире возрастает влияние программных систем на жизнь и здоровье людей. Бортовые программные системы отвечают за критически значимые функции. Они управляют автомобилями, кораблями, электростанциями и т.д. Такие системы обычно имеют распределенный характер. Они состоят из нескольких подсистем, каждая из которых работает независимо по своему алгоритму, взаимодействуя с другими подсистемами для выполнения общих целей и задач.

Известно, что в распределенных системах трудно выявить ошибки и понять причину их возникновения. Исправление ошибок в реализации бортовых систем, построенных по традиционной технологии, требует значительных затрат: приходится возвращаться к этапу проектирования и повторять проверку модифицированной версии программы вновь, см., например, [1].

Если в основу разработки программы закладывать модель, то появляется возможность включения в процесс проектирования методов верификации. Верификация модели позволяет обнаруживать большинство логических ошибок на ранних стадиях разработки. Эта же модель может быть впоследствии использована для

генерации кода управляющего ядра системы. Такой подход называется проектированием под управлением моделей (Model-Driven Engineering — MDE) [2].

В работе предлагается конкретная реализация MDE на основе выбранных конструкций языка UML. В качестве метода верификации используется model checking [3]. Model checking (проверка модели) — это автоматический формальный метод проверки того, что заданное свойство, сформулированное на языке темпоральной логики, истинно на структуре, представляющей модель разрабатываемой программной системы. При формальной проверке модели проводится анализ всех возможных вариантов поведения системы, что позволяет разработчику гарантировать корректность проектируемой системы относительно заданного набора свойств.

Разработанная методика тестировалась на хорошо известных протоколах — альтернирующего бита, взаимного исключения, обедающих философов. В качестве основного примера, демонстрирующего применимость методики к практическим задачам, в статье рассматривается реальная бортовая система управления энергоснабжением судна. Эта система управления изначально реализована по технологии RUP (Rational Unified Process) [4] независимым коллективом разработчиков, которые ее тщательно протестировали и сдали заказчику. По коду программы был проведен реинжиниринг для построения той модели, с которой, согласно выбранной методике, и следовало бы начинать разработку. В результате верификации модели обнаружен ряд критических ошибок, которые подтверждаются на трассах кода.

2. Язык описания моделей

Бортовая система управления обычно состоит из нескольких подсистем. Часть внутренних алгоритмов, а иногда и целые подсистемы, не имеют непосредственного отношения к логике взаимодействия, потому не требуют верификации методом проверки модели. Корректность алгоритмов и подсистем, не связанных с взаимодействием, может проверяться другими методами обеспечения качества, например, статическим анализом, тестированием.

Независимые взаимодействующие подсистемы, определяющие логику управления всей системой как единым целым, составляют ядро управления. Ядро управления принадлежит классу реагирующих систем, т.е. систем, дающих отклик на внешние события в зависимости от своего состояния. К реагирующим системам относятся операционные системы, протоколы коммуникации, планировщики, контроллеры, параллельные взаимодействующие программы, драйверы устройств.

Разработанная методика основана на построении и верификации именно модели ядра бортовой системы управления.

Гибкий, платформенно-независимый язык моделирования общего назначения UML (Unified Modeling Language) хорошо знаком архитекторам программного обеспечения и широко применяется в разных прикладных областях. UML, как универсальный язык моделирования, включает множество конструкций, которые не являются необходимыми для описания ядра логических систем управления. Диаграммы и конструкции языка UML для методики выбирались, исходя из ограничений, накладываемых на создаваемые модели: рассматриваются модели с конечным числом

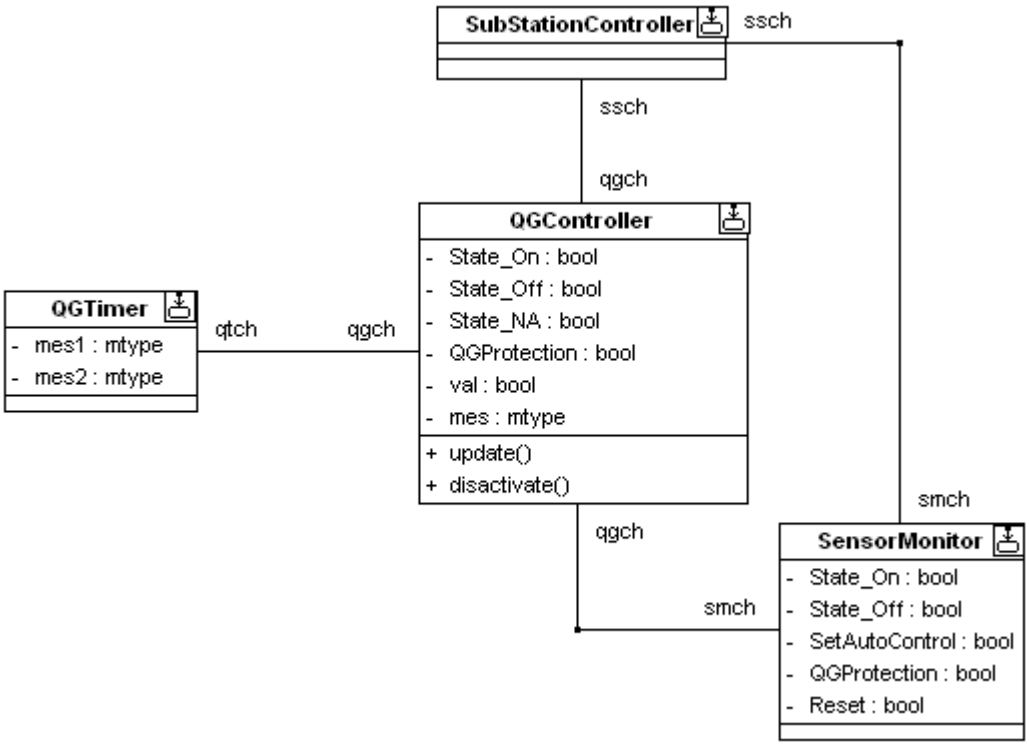

Рис. 1. Пример диаграммы классов СУЭС

состояний; глобальное поведение системы складывается из интерливинга (чередования) шагов поведения независимых подсистем и их взаимодействия; взаимодействие между подсистемами происходит по асинхронным каналам конечной емкости.

Поведение реагирующих систем удобно и компактно описывают UML диаграммы состояний (UML state machine). Статические связи между элементами системы задаются UML диаграммами классов (UML class diagram). Конструкции этих UML диаграмм позволяют, в принципе, определить более сложные структуры и поведение, чем в моделях с конечным числом состояний, например, динамическое создание и удаление объектов, неограниченные очереди событий, переменные с неограниченной областью определения. Однако в предлагаемой методике такие конструкции не используются, т.к. мы ограничиваемся моделями с конечным числом состояний.

Рассмотрим фрагмент бортовой системы управления энергоснабжением судна (СУЭС) для демонстрации подъязыка UML, используемого для создания моделей. Эта бортовая система контролирует поведение двух судовых подстанций. Каждая подстанция состоит из дизеля, генератора и генераторного автомата. Управление обрабатывает команды оператора и сигналы от датчиков. Показания датчиков меняются в зависимости от потребления энергии первичными и вторичными потребителями, давления масла, температуры приборов и других внешних условий.

UML-модель бортовой СУЭ была построена вручную в среде BoUML [5] на основе программной реализации системы управления (C ++). При разработке модели независимость подсистем сохранена. Каждая независимая подсистема в UML-модели имеет свою диаграмму состояний, отражающую ее поведение. Каналы обме-

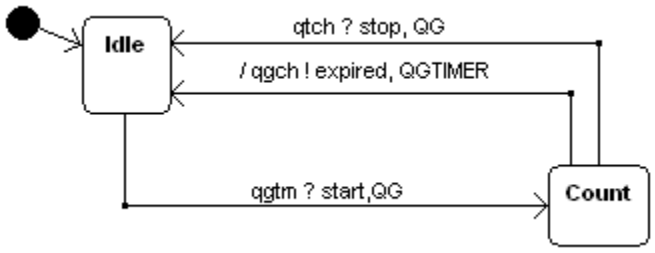

Рис. 2. UML диаграмма состояния таймера ГА

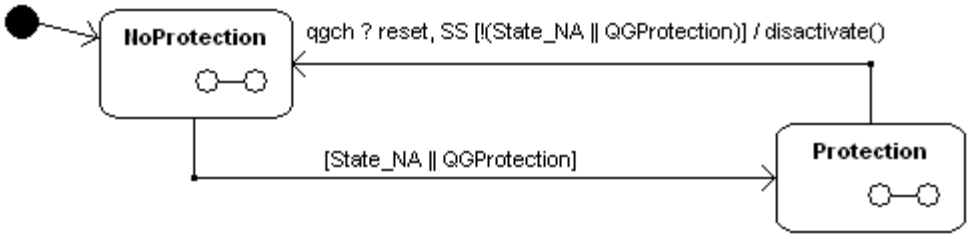

Рис. 3. UML диаграмма состояний контроллера генераторного автомата

на сообщениями, формат сообщений, простейшие функции, локальные переменные подсистем также соответствуют реализации. Глубина каналов снижена для уменьшения размера модели. Основные временные затраты при моделировании определялись разработкой гиперсостояний независимых подсистем, т.к. это потребовало дополнительной структуризации исходных алгоритмов реализации. Критерием адекватности построенной для этого примера модели служит подтверждение результатов ее верификации в исходном коде.

На рис. 1 приведена диаграмма классов, соответствующая фрагменту подстанции СУЭС. Центральным объектом в этом примере является контроллер генераторного автомата `QGController`. Контроллер генераторного автомата, взаимодействуя с подсистемой опроса и установки датчиков `SensorMonitor`, определяет состояние генераторного автомата (ГА) по значениям соответствующих датчиков (`State_On`, `State_Off`, `QG_Protection`). Для нашего примера достаточно знать, что контроллер подстанции `SubStationController` отвечает за взаимодействие с оператором: он получает сигнал о переходе в автоматический режим управления `SetAutoControl` и сигнал о сбросе защиты `Reset`. Основная задача при верификации этого фрагмента состоит в обнаружении и реакции на события защиты.

Каждому из приведенных на рисунке объектов соответствует своя карта состояний (statechart). Карта состояний соответствует формальной спецификации всех состояний объекта и возможных переходов между состояниями. При этом выделяются управляющие состояния объекта, локальные и глобальные переменные. Управляющее состояние описывает множество реальных состояний системы, отличающихся в нем набором значений переменных. Переходы диаграмм состояний помечены конструкцией: событие [охранное условие перехода]/набор действий. Охранное условие перехода представляет собой предикат от значений переменных. В качестве отдель-

ного действия на переходе разрешены либо присваивание, либо отправка сообщений. Для того чтобы сократить запись часто повторяющихся наборов действий используются методы, например, `disactivate()` (рис. 3). В данном подходе методы могут содержать только набор действий. Любая из частей этой конструкции может быть пустой.

На рис. 2 приведена карта состояний таймера ГА `QGTimer`. Таймер запускается в некоторых переходных режимах работы ГА. Из своего активного состояния `Count` таймер выходит недетерминированно: либо по получении сообщения от контроллера ГА, либо по истечении таймера, что сопровождается отправкой сообщения ГА.

Контроллер ГА `QGController` имеет достаточно сложное поведение, которое задается диаграммой состояний с гиперсостояниями. Гиперсостояние объединяет несколько состояний, имеющих идентичную реакцию на некоторый набор событий. На рис. 3 приведена диаграмма состояний контроллера, соответствующая поведению ГА при обработке события срабатывания защиты. В случае срабатывания защиты или неопределенного состояния ГА контроллер переходит в гиперсостояние защиты, из которого может выйти только по команде оператора `reset`.

Использование карт состояний, управляющих состояний, методов позволяет компактно описывать большие модели поведения объектов, исходя из технического задания на разрабатываемую систему логического управления.

3. Трансляция моделей в язык Promela

Для верификации методом проверки модели используется программное средство SPIN, разработанное в исследовательском центре Bell Labs [6]. SPIN позволяет верифицировать свойства, выраженные на языке линейной темпоральной логики LTL, строя синхронную композицию модели и автомата Бюхи, соответствующего отрицанию свойства. Модели, верифицируемые SPIN, должны быть описаны на языке Promela.

В предлагаемой методике разработан транслятор UML-моделей в Promela. Входным языком транслятора является XML. Графический редактор BoUML предоставляет возможность экспорта UML-моделей в XML по стандарту обмена метамоделями XMI 2.1 (XML Metadata Interchange). Этот стандарт устанавливает правила преобразования метамоделей на языке XML, т.е. определяет, как использовать XML-теги для представления моделей. При разработке транслятора для упрощения части операций использовалась XSLT-трансляция [7].

Каждый из объектов системы, имеющих независимое поведение, транслируется в отдельный процесс Promela. Каналы обмена сообщениями между объектами, заданные на диаграмме классов (рис. 1), объявляются глобально. Методы переводятся в конструкции Promela `inline`; так метод ГА `update()` (рис. 1) транслируется в код на Promela:

```
inline QGControllerupdate()
{
  smch ! state_on, QG;
  smch ! state_off, QG;
```

```
  smch ! qgprotection, QG;
}
```

Алгоритм поведения независимого объекта, построенный в соответствующей диаграмме состояний (рис. 2), включается в тело процесса:

```
/* Объявление процесса с именем QGTimer */
proctype QGTimer()
{
/* Объявления локальных переменных */
  int _state = QGTimerIdle;
  mtype mes1, mes2;
/* Бесконечный цикл по состояниям */
  do
/* Проверка текущего состояния */
  :: (_state == QGTimerIdle) -> /* обработка переходов Idle */
  :: (_state == QGTimerCount) -> /* обработка переходов Count */
  od
}
```

При обработке исходящих переходов из текущего состояния допускается деятельность при входе и выходе из состояния, оформленная в соответствующих методах `entry()`, `exit()`. Эти методы должны содержать только присваивания переменных.

```
  :: (_state == QGTimerCount) ->
      if
      /* Если входной канал непустой */
      :: nempty(qtch) ->
         /* то считывается сообщение из него*/
          qtch ? mes1, mes2;
          if
          /* Если сообщение - событие для перехода в другое состояние,*/
          :: (mes1 == stop) && (mes2 == QG) ->
              /* выполняются действия на выходе из текущего состояния,
              например, QGTimerexit(QGTimerCount);*/
              /* выполняются  действия при переходе, если они имеются*/
              _state = QGTimerIdle;/* меняется состояние*/
              /* выполняются действия при входе в целевое состояние
              QGTimerentry(QGTimerIdle);*/
              /* Все другие сообщения отбрасывается*/
          :: else -> skip
          fi
      /* Недетерминированно может истечь таймер */
      :: true ->
         qgch ! expired, QGTIMER;
         _state = QGTimerIdle;
```

```
fi
```

Стандартное для UML 2.0 [8] ограничение run-to-completion (выполнение до завершения) обеспечивается следующим образом: проверка сообщений в канале осуществляется только после завершения всех действий, связанных с переходом в состояние, включая действия на выходе из состояния, действия на переходе, действия при входе в новое состояние.

Карта состояний с гиперсостояниями разворачивается в эквивалентную плоскую, т.е. не содержащую гиперсостояний. Преобразование является стандартным: для тех переходов, где гиперсостояние, например, `Protection`, является целевым, оно заменяется начальным состоянием своего вложенного набора состояний. Вместо тех переходов, где гиперсостояние `Protection` является источником, строится по переходу из каждого его вложенного состояния.

Верификация методом проверки модели проводится для замкнутых систем: кроме ядра управления требуется моделировать поведение среды, порождающей события, например, поведение оператора за пультом. Поведение среды можно ограничить, задав сценарий, т.е. конечную фиксированную последовательность событий. В этом случае модель среды соответствует поведению среды, согласующемуся с каким-либо вариантом ее использования. Однако такой подход не предполагает появления случайного, не предусматриваемого стандартным сценарием события, например, внештатного нажатия кнопки включения противопожарной системы на подводной лодке, а потому не позволяет выявить тех поведений ядра управления, которые никогда не должны происходить. В предлагаемой методике предусмотрена возможность моделирования среды недетерминированной картой состояний, которой покрываются все возможные сценарии поведения среды.

4. Типы проверяемых требований

При помощи верификации невозможно гарантировать, что система корректна относительно всех возможных требований, потому что пространство требований является открытым множеством. Структуризация спецификации бортовой системы управления по описываемой ею функциональности позволяет выделить наиболее значимые типы свойств. Предлагается рассматривать четыре типа (множества) требований корректности, выполнение которых на модели свидетельствует о достаточно высоком качестве окончательной системы:

1. Требования корректности, связанные с отсутствием типичных ошибок параллельных процессов, так называемые базовые требования (возможность возврата в начальное состояние, отсутствие блокировки процессов и одновременного доступа к критическим интервалам и некоторые другие требования).

2. Требования управляемости системы (достижимость всех состояний, характеризующих выполнение требуемых функций).

3. Требования надежности (выполнение критических и опасных режимов, переход в такие режимы только согласно предписанным регламентом процедурам).

4. Требования корректности, вытекающие из технического описания функционирования системы (требования к поведению системы во всех базовых сценариях и режимах функционирования).

Данная структуризация исходит из потребностей разработчиков бортовых систем. По формальным характеристикам предложенные типы свойств могут пересекаться.

5. Верификации системы управления энергоснабжением судна

При верификации модели СУЭС был найден ряд некорректностей. Перечислим некоторые из них. Одной из часто встречающихся некорректностей распределенных систем является взаимная блокировка процессов. В построенной модели были обнаружены блокировки процессов, которые возникают из-за переполнения каналов обмена. Поскольку размер модели не позволяет проверить базовые свойства на объемах канала, использовавшихся в реализации, то можно было бы отнести данную ошибку к ограничениям, возникающим при моделировании. Однако выявление данной ошибки позволило обнаружить в исходной реализации большое количество избыточных с логической и алгоритмической точки зрения взаимодействий между процессами. Данную проблему удается решить удалением лишних пересылок.

Каждый из режимов поведения реальной системы управления характеризуется определенным состоянием датчиков. Как только система управления получает сообщение от среды о том, что состояние какого-либо датчика не соответствует ее текущему режиму, по технической спецификации она должна поменять режим работы. В частности, автоматический активный режим генераторного автомата (ГА) характеризуется непротиворечивым состоянием датчика ГА и отсутствием условий срабатывания защиты. Это требование к поведению системы формально записывается следующим образом: "проверить, что на всех трассах выполняется следующая LTL формула $G!(q \;\&r\; \&\; post)$", где q — метка активного режима, r — локальное значение датчика, соответствующее противоречивому состоянию ГА, $post$ — контроллер ГА после обработки сообщения. При проверке этого свойства верификатор обнаружил контрпример, в котором предъявленное требование нарушается на 598 шаге вычисления (рис. 4). Эта некорректность действительно возникала в программе и вызывалась отсутствием обработки противоречивого состояния датчика генераторного автомата и была исправлена добавлением дополнительного перехода из активного режима.

В технической спецификации СУЭ утверждается, что выход из режима защиты генераторного автомата возможен только при нажатии кнопки "Сброс защиты". Для проверки этого требования было сформулировано следующее свойство на языке LTL: $F\ q \;\Rightarrow\; !q\ U\ p$, где p — утверждение о том, что кнопка "сброс защиты" нажата, а q — утверждение о том, что система вышла из режима защиты. Проверяемое свойство гарантирует, что никогда не будет так, что система вышла из режима защиты без нажатия кнопки сброса. При верификации этого свойства на модели обнаружена трасса, на которой это свойство не выполняется. Анализ по-

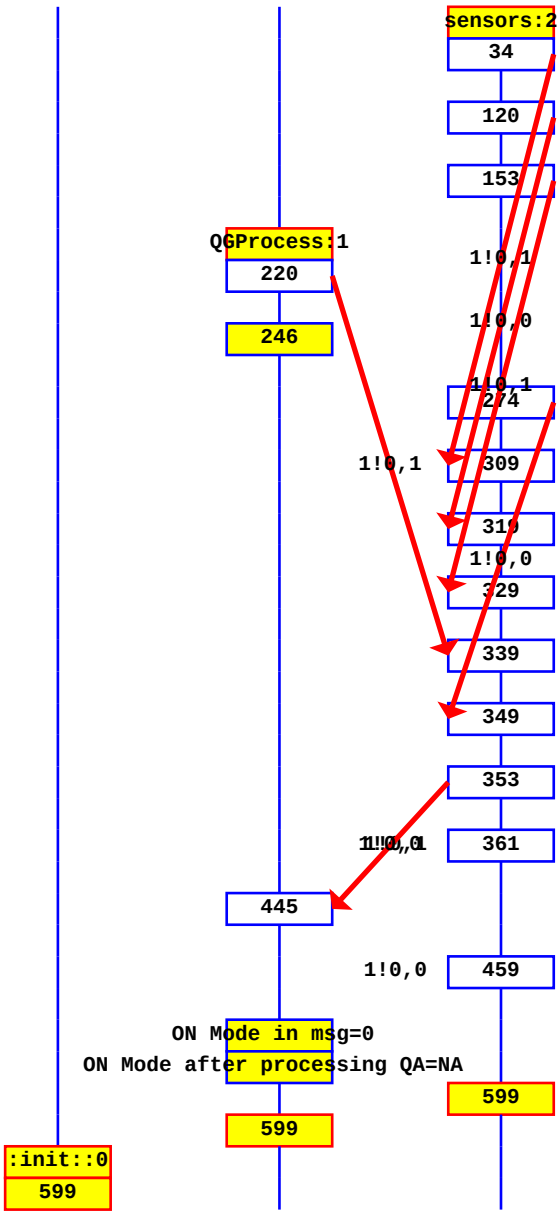

Рис. 4. Нарушение условия нахождения ГА в активном режиме, обнаруженное системой SPIN при верификации

казал, что эта некорректность объясняется отсутствием в исходной программе на C ++ переходного режима; ошибка была исправлена его добавлением.

Найденные при верификации построенной модели ошибки были подтверждены тестированием исходной реализации кода на C++. Их исправление в реализации в некоторых случаях приводит к значительной модификации кода.

При моделировании трудно оценить, к чему приводят на физическом уровне выявленные логические ошибки. Например, потеря управления может быть результатом блокировки процессов. Параллельно и независимо с разработкой и верификацией модели проводились ходовые испытания судна, на котором была установлена исходная реализация СУЭС. При ходовых испытаниях было выявлено несколько ошибок, в частности, ошибка блокировки управления, когда система управления неожиданно перестала отвечать на команды и отключила энергоснабжение. В результате остановились оба двигателя, судно загородило фарватер.

Предложенная методика позволяет выявить подобные ошибки еще на этапе проектирования, следовательно, в окончательной системе таких бы ошибок не появилось.

6. Выводы

Несмотря на то, что общая концепция, формальное обоснование, базовые алгоритмы верификации моделей программ методом model checking разработаны, их использование в технологии проектирования конкретных классов программных систем требует дополнительных исследований. Выбор языка описания моделей, формирование типов свойств, подлежащих проверке, решение возникающей проблемы "взрыва числа состояний" при верификации реальных практических систем остаются актуальными задачами. Работы по разработке корпоративных технологий проектирования с поддержкой верификации проводятся некоторыми корпорациями и организациями за рубежом (Rockwell Collins, NASA, Microsoft). Однако результаты этих работ, как правило, недоступны. В нашей стране подобные исследования ведутся в основном научными организациями. Авторы работы [9] предлагают явно выделять состояния при построении моделей, в частности, дизель-генератора. Такой подход хорош только для небольших учебных моделей, т.к. в больших системах он чреват дополнительными ошибками. В работе [10] UML-моделирование на основе диаграмм активностей успешно использовалось, но не для верификации, а для автоматической генерации тестов.

В данной работе предложено использовать проектирование на основе моделей (Model-Driven Engineering) для класса систем логического управления. Для описания моделей выделен подъязык UML, разработан транслятор в Promela и даны рекомендации по разбиению проверяемых свойств на классы, вытекающие из технических и эксплуатационных требований бортовой системы управления. Полученные результаты и опыт эксплуатации на примере системы управления энергоснабжением судна показал эффективность разработанной методики. В результате верификации было обнаружено несколько тонких ошибок, оставшихся в ПО после его тестирования. Дальнейшие исследования могут быть направлены на разработку алгоритмов

и методов автоматического снижения размера модели, т.е. построения абстракций, достаточных для верификации конкретных свойств. Другое не освещенное здесь направление работы в рамках предложенной технологии — эффективная генерация автоматов Бюхи при помощи альтернирующих автоматов.

Список литературы

1. Royce W. W. Managing the Development of Large Software Systems // Proceedings of the 9th International Software Engineering Conference. Computer Society Press, 1987. С. 328–338.

2. Model Driven Architecture (MDA). ormsc/2001-07-01: OMG, 2001.

3. Карпов Ю.Г. Model Checking. Верификация параллельных и распределенных программных систем. СПб: БХВ-Петербург, 2010.

4. Kruchten P. The Rational Unified Process: An Introduction. Addison-Wesley Longman Publishing Co., Inc. 2003.

5. Pages B. BoUML user manual. 2010. http:/bouml.free.fr.

6. Holzmann G. Spin Model Checker, The Primer and Reference Manual. Addison Wesley, 2003.

7. Валиков А.Н. Технология XSLT. СПб.: БХВ-Петербург, 2002.

8. OMG Unified Modeling Language (OMG UML), Superstructure, v2.1.2. Object Management Group (OMG), 2007. http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF.

9. Шалыто А.А., Туккель Н.И. Проектирование программного обеспечения системы управления дизель-генераторами на основе автоматного подхода // Системы управления и обработки информации. СПб.: ФГУП «НПО «Аврора», 2003. Вып. 5. С. 62–82.

10. Калинов А.Я., Косачев А.С., Посыпкин М.А., Соколов А.А. Автоматическая генерация тестов для графического пользовательского интерфейса по UML-диаграммам действий // Труды института системного программирования РАН. М., 2004. Т. 8, ч. 1. С. 99–117,

Distributed embedded control systems design with verification support

Shoshmina I.V.

Keywords: Model-Driven Engineering, model checking, UML, LTL, Promela

We consider a problem of integrating a formal method of verification (model checking) into the process of designing complex distributed software systems to improve the quality of software. We use an approach based on the Model-Driven Engineering. It allows us to structur the design process by selecting and verifying a system core, consisting of independent subsystems and being responsible for logical management of the entire system as a whole. The proposed method was tested on a real control system of vessel power supply.

Сведения об авторе:
Шошмина Ирина Владимировна,
Санкт-Петербургский государственный политехнический университет,
факультет технической кибернетики,
кафедра распределенных вычислений и компьютерных сетей,
старший преподаватель