Применение UniTesK к тестированию встроенных систем

Н.В. Пакулин (npak@ispras.ru)

Аннотация. В статье обсуждаются вопросы применимости технологии тестирования UniTesK к функциональному тестированию программного обеспечения встроенных систем на примере сенсорных сетей под управлением TinyOS. В работе выделены сходство и отличие указанного класса ПО от систем, для которых хорошо разработаны методики применения UniTesK. Представлены результаты опытного проекта по исследованию применимости CTesK к тестированию ПО под управлением TinyOS.

1. Введение

В статье рассматриваются вопросы тестирования встроенных систем с очень ограниченными ресурсами, прежде всего с небольшим объёмом оперативной памяти. Практически все доступные ресурсы используются для поддержания работы приложения на устройстве. Ограниченные ресурсы делают невозможным развёртывание тестовой системы на устройстве, поэтому существующие технологии тестирования необходимо адаптировать для тестирования встроенного программного обеспечения.

В данной работе вопросы тестирования встроенного программного обеспечения рассматриваются на примере тестирования приложений для сенсорных сетей под управлением TinyOS [1]. TinyOS используется для программирования устройств, снабжённых датчиками (температуры, влажности, освещённости и т.п.) и маломощным коротковолновым приёмо-передатчиком для периодической передачи показаний датчиков.

TinyOS служит хорошим примером современных приложений встроенных систем с очень ограниченными ресурсами. Объём оперативной памяти, доступной типичному устройству TinyOS, составляет 4-16 Кбайт.

В статье обсуждаются возможности применения технологии тестирования UniTesK 2 к разработке тестовых наборов для программного обеспечения сенсорных сетей под управлением TinyOS. Указанный класс программного обеспечения имеет много общего с теми системами, для которых применимость UniTesK хорошо исследована – программными интерфейсами, реализациями телекоммуникационных протоколов. В статье показано, как перенести опыт тестирования программных интерфейсов и реализаций протоколов на задачу тестирования TinyOS, и как адаптировать UniTesK к

особенностям TinyOS; тем самым доказывается применимость UniTesK к тестированию указанного класса ПО. Вывод подкрепляется результатами пилотного проекта по тестированию ПО под управлением TinyOS средствами CTesK [3] – реализации UniTesK для языка C.

Статья построена следующим образом. Раздел 2 содержит краткое введение в TinyOS. В разделе 3 дано общее описание метода разработки тестовых наборов UniTesK. Раздел 4 посвящен применению UniTesK к тестированию встроенных систем. В разделе 5 приведены результаты пилотного проекта по тестированию компонента Attr. В разделе 6 обсуждается вопрос применимости формальных спецификаций UniTesK к аналитической верификации встроенного ПО. Раздел 7 – Заключение.

2. Краткий обзор TinyOS

TinyOS — это операционная система, разработанная для сетевых встроенных приложений. Программная модель TinyOS адаптирована для приложений, основанных на событиях, и отличается очень небольшим объёмом занимаемой памяти (базовые функции ОС занимают порядка 400 байт, включая данные и код). В качестве важнейших особенностей TinyOS можно выделить:

- компонентную архитектуру;
- простую модель параллельности, основанную на событиях;
- расщеплённые операции;
- специализированный язык разработки.

Компоненты TinyOS и приложения разрабатываются на языке nesC [4], [5] – специализированном расширении языка C, которое поддерживает указанные особенности TinyOS.

2.1. Компонентная архитектура TinyOS

Компонентная архитектура TinyOS предоставляет набор повторно используемых компонентов. Компоненты собираются в приложение посредством спецификаций связывания интерфейсов; спецификации связывания не зависят от реализации компонентов. В TinyOS различают два вида компонентов: модули и конфигурации. Модули содержат состояние и операции. Конфигурации представляют собой наборы модулей и конфигураций, связанные в соответствии со спецификацией связывания. Приложение можно рассматривать как наибольшую конфигурацию, развёрнутую на устройстве.

Благодаря разделению функций TinyOS на отдельные компоненты разработчик приложения может использовать только те компоненты, которые нужны ему для решения задачи на заданной аппаратуре, тем самым минимизируя затраты ресурсов на устройстве.

Большая часть компонентов, поставляемых с TinyOS, представляет собой программные модули. Интерфейсы доступа к аппаратуре также имеют вид компонентов и с точки зрения разработчика приложения неотличимы от обычных программных модулей. Концепция компонентов отражена в языке nesC.

2.2. Параллельность, события и задачи

В TinyOS различаются два контекста исполнения — события и задачи. Задачи представляют собой способ отложенного выполнения вычислений. События соответствуют асинхронным операциям, преимущественно прерываниям аппаратуры.

Выполнение задачи не может быть прервано выполнением другой задачи (но может быть прервано событием). Выполнение задач управляется планировщиком. Компоненты могут ставить задачи в очередь, планировщик запускает выполнение задач в моменты времени, когда не обрабатываются события.

Обработка событий может прерываться обработчиками других событий, но не может прерываться переключением на исполнение задач. Для обеспечения достаточной степени реактивности приложения продолжительные вычисления, которые могут возникнуть при обработке событий, рекомендуется ставить в очередь задач.

2.2.1. Расщеплённые операции

В силу того, что задачи выполняются без переключения на другие задачи, в TinyOS рекомендуют разделять операции на две — запрос и завершение. Так реализован доступ к большинству аппаратных функций. Запрос возвращает управление почти мгновенно, завершение операции сигнализируется событием[1].

Примером расщеплённой операции может служить отправка пакета в сеть: компонент запускает передачу сообщения по радиосети, по завершении передачи компонент получает событие sendDone.

Проблема конкуренции за ресурс, как правило, решается путем явного запрещения параллельных запросов. В примере с отправкой сообщений, в том случае, если радиопротокол не поддерживает параллельную отправку сообщений (например, в разных частотных диапазонах), запросы на отправку сообщений будут отклоняться до окончания текущей передачи[2].

Простая модель параллельности в TinyOS позволяет реализовать параллельные вычисления с небольшими накладными расходами, если сравнивать с многопотоковой параллельностью, в которой драгоценная память тратится на стек заблокированных нитей, ожидающих доступ к занятому устройству. Тем

[1] Ряд быстрых операций, например, переключение светодиодов, выполняется по запросу целиком, без события завершения.

[2] Разумеется, могут быть альтернативные реализации протоколов, которые поддерживают очереди исходящих сообщений.

не менее, в TinyOS также возможны различные ошибки, характерные для параллельного исполнения, включая тупиковые ситуации, конфликты при доступе к памяти.

2.2.2. Атомарность

В nesC рассматриваются два контекста исполнения — синхронный и асинхронный. Синхронный контекст соответствует выполнению задач TinyOS, асинхронный – обработке событий в TinyOS. Так как выполнение задач в TinyOS не прерывается другими задачами, то синхронный код считается атомарным по отношению к синхронному коду. Для задания атомарных участков в асинхронном коде есть специальная синтаксическая структура atomic.

2.3. Язык nesC

Язык nesC отражает основные концепции TinyOS. Язык основан на концепции компонентов и естественным образом реализует модель событий и задач.

Язык nesC реализован как расширение языка C. В него добавлены синтаксические конструкции для описания компонентов TinyOS, событий и доступа к переменным состояния.

В языке nesC допустимы только статические конструкции, отсутствуют динамическое выделение памяти и указатели на функции. Компонентная модель nesC и параметризация компонентов покрывают большинство ситуаций, для которых в традиционном C требуется использование динамической памяти и указателей на функции.

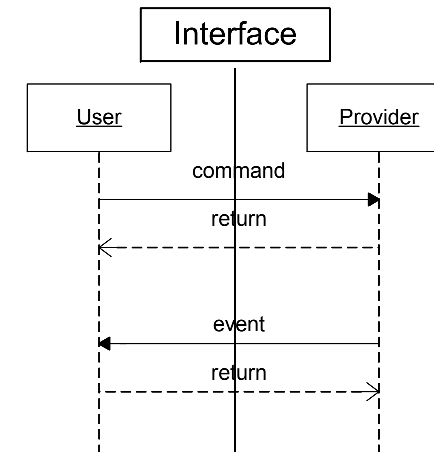

Рис. 1. Интерфейс в nesC.

Приложение nesC собирается из компонентов. В языке nesC различаются интерфейсы и реализация компонентов. Компоненты предоставляют и

используют интерфейсы. Единственный способ обратиться к компоненту — вызвать интерфейсный метод компонента.

Интерфейс nesC состоит из спецификаций методов. Методы делятся на две группы: команды и события. Команды соответствуют вызовам компонента со стороны окружения, события обозначают вызовы окружения из компонента. Это проиллюстрировано на рисунке 1.

Различие между предоставлением интерфейса и использованием интерфейса заключается в том, что поставщик интерфейса реализует команды, а пользователь интерфейса реализует события.

```
interface ADC {
     command result_t getData();
     event result_t dataReady(uint16_t data);
}
```

Рис. 2. Пример интерфейсного типа.

В языке nesC предусмотрена концепция интерфейсного типа. Интерфейсный тип позволяет компактно записывать одинаковые интерфейсы.

Интерфейс на рисунке 2 является примером представления расщеплённой операции в языке nesC. Этот интерфейс предоставляет компоненты, которые осуществляют сбор данных с датчиков. Команда getData инициирует опрос датчика и возвращает управление. Готовые данные будут доставлены как параметр data события dataReady.

Двунаправленные интерфейсы удобны для описания обработчиков аппаратных прерываний. Для сравнения, в языках с однонаправленными процедурными интерфейсами требуется либо постоянно опрашивать аппаратуру, либо разделять интерфейс на два — операции с аппаратурой и соответствующие прерывания.

2.3.1. Реализация компонентов в языке nesC

В языке nesC поддерживаются два вида компонентов — модули и конфигурации. Модули содержат программный код и реализуют один или несколько интерфейсов. Конфигурации используются для связывания других компонентов в одно целое, соединяя компоненты в соответствии с правилами связывания. Каждое приложение nesC описывается некоторой конфигурацией верхнего уровня, которая связывает все используемые компоненты. Язык nesC предоставляет богатый набор правил связывания. Можно связывать:

- одну или несколько команд с одной или несколькими командами, одно или несколько событий с одним или несколькими событиями;
- один или несколько интерфейсов с одним или несколькими интерфейсами одного и того же типа.

Для TinyOS уже разработаны несколько наборов компонентов, предназначенных для создания специализированных приложений. Один из

таких наборов, TinyDB [6], позволяет организовать доступ к показаниям сенсоров на устройствах в сети сенсоров в виде операций чтения/записи в некоторой виртуальной базе данных.

3. Краткий обзор UniTesK

Данный раздел содержит беглое введение в UniTesK. Более подробное описание можно найти в [2]. С 1994 года в ИСП РАН активно разрабатываются методы и инструменты тестирования программного обеспечения на основе формальных методов.

В 1994-1999 годах по контрактам с Nortel Networks в ИСП РАН был разработан и активно использовался метод тестирования KVEST [7]. Этот метод отличался от обычных методов тестирования индустриального программного обеспечения тем, что в KVEST использовались формальные спецификации в форме пред- и постусловий для построения тестовых оракулов, а также модель конечных автоматов для построения тестовых воздействий.

Применение KVEST показало, что формальные методы можно с большим успехом применять при тестировании индустриального программного обеспечения. Опыт применения KVEST также показал, что использование академических языков формальных спецификаций и специальных языков описания тестов препятствует встраиванию инструментов, основанных на этих языках, в процесс разработки ПО. Чем ближе язык спецификаций к языку, на котором ведется разработка, тем проще разработчикам писать спецификации и тесты [8].

UniTesK разрабатывался на основе опыта, полученного при разработке и применении KVEST. Рассмотрим основные особенности UniTesK:

- Разделение построения тестовых воздействий и проверки правильности поведения целевой системы. Тестовые воздействия строятся в *тестовых сценариях,* а проверка правильности поведения целевой системы производится в *тестовых оракулах*;
- Автоматизированное построение тестовых воздействий;
- Представление функциональных требований к целевой системе в виде формальных спецификаций;
- Для записи формальных спецификаций используется язык, «близкий» к языку, на котором разработана целевая система;
- Автоматическая генерация тестовых оракулов из спецификаций;
- Оракулы и реализация связаны посредством тонкой прослойки *медиаторов;*
- Язык описания тестовых воздействий «близок» к языку, на котором разработана целевая система;
- Автоматически генерируются критерии качества покрытия требований;

- Автоматически производится оценка качества покрытия требований при прогоне тестов.

3.1. Оракулы и спецификации

Оракулы – это процедуры, которые проверяют выполнение ограничений, наложенных на поведение целевой системы. В UniTesK оракулы полностью автоматически генерируются из описаний ограничений.

В UniTesK ограничения описываются преимущественно в форме имплицитных спецификаций стимулов и реакций. Кроме того, часть ограничений представляется в виде ограничений на значения типов (инварианты типов) и ограничений на значения глобальных переменных (инварианты глобальных переменных).

3.1.1. Стимулы и реакции

В UniTesK различаются два вида операций, на которые накладываются ограничения, – стимулы и реакции.

Стимул – это воздействие, которое совершается окружением над системой. Реакция – это воздействие со стороны системы на окружение.

3.1.2. Спецификация

Имплицитные спецификации стимулов и реакций состоят из пред- и постусловий. Предусловие для стимула содержит требования к тому, в каком состоянии и с какими параметрами можно оказывать воздействие на целевую систему. Предусловие для реакции задаёт ограничения на состояния, в которых система может демонстрировать реакции.

После воздействия целевая система может продемонстрировать реакции и/или перейти в другое состояние. Постусловие для стимула определяет, допустимо ли изменение состояния, а постусловия для реакций определяют допустимость продемонстрированного поведения целевой системы.

Состояние целевой системы моделируется набором структур данных, называемых абстрактным состоянием. В пред- и постусловиях используется информация, содержащаяся в абстрактном состоянии, для вынесения вердикта.

3.2. Тестовые сценарии

Тестовый сценарий в UniTesK определяет последовательность воздействий, которые оказываются на целевую систему. В качестве теоретической основы для построения тестового сценария в UniTesK выбрана модель конечных автоматов.

Тестовый сценарий обладает собственным состоянием, которое, как правило, вычисляется на основе состояния целевой системы. В каждом состоянии задаются воздействия, которые в данном состоянии можно оказать на целевую

систему. Тестовый сценарий в процессе работы обходит все состояния и в каждом состоянии оказывает все перечисленные воздействия.

На рисунке 3 изображен пример тестового сценария, в котором насчитывается три состояния (*S1*, *S2* и *S3*) и семь воздействий (*I1_1*, *I1_2*, *I2_1*, *I2_2*, *I3_1*, *I3_2*, *I3_3*). При тестировании тестовый сценарий побывает в каждом состоянии и пройдет по каждой дуге.

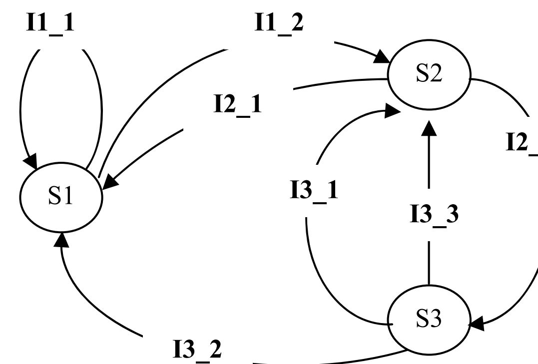

Рис. 3. Пример автомата с тремя состояниями и семью переходами.

К сожалению, для большинства целевых систем невозможно получить описание тестового сценария из спецификаций полностью автоматически, без помощи человека. Можно указать следующие причины:

- Число состояний целевой системы, как правило, очень велико;
- Число воздействий, которые можно оказать на целевую систему в каждом состоянии, как правило, тоже очень велико.

Вместе с тем, число групп разных состояний, как правило, вполне обозримо. Но невозможно автоматически определить критерий различения состояний, здесь нужна помощь человека – разработчика тестов.

Число различных воздействий на целевую систему также много меньше общего числа воздействий. Из формальных спецификаций можно автоматически получить критерий различения воздействий, но автоматически построить воздействия можно только в простейших случаях.

В UniTesK реализован компромиссный подход к разработке тестовых сценариев. Разработчик тестового сценария пишет процедуру различения состояний целевой системы, тем самым определяя классы эквивалентности состояний целевой системы. Каждый класс эквивалентности определяет одно состояние автомата тестового сценария.

Разработчик тестового сценария также задает процедуру, которая строит всевозможные тестовые воздействия, то есть переходы автомата тестового сце-

нария. Эти воздействия фильтруются в соответствии с одним из сгенерированных критериев покрытия. Фильтры отсеивают избыточные воздействия, то есть воздействия, не улучшающие уже достигнутого покрытия. Фильтрация воздействий существенно упрощает написание процедур перебора воздействий.

По предоставленным описаниям динамически строится граф состояний автома-та тестового сценария. При обходе графа автоматически отслеживается покры-тие требований, описанных в спецификациях для генерации отчета о покрытии.

3.3. Медиаторы

Под медиаторами в UniTesK понимается промежуточный слой между оракулами и реализацией. Необходимость введения медиаторов вызвана тем, что в UniTesK воздействия на целевую систему и реакции целевой системы описываются в терминах модели, содержащейся в спецификациях. Перед тем, как оказать воздействие на целевую систему, необходимо перевести параметры воздействия из модельного представления в представление реализации, а после того, как воздействие оказано, необходимо перевести реакции целевой системы в модельное представление. Также необходимо отобразить изменения состояния целевой системы в абстрактном состоянии.

В UniTesK медиаторы осуществляют необходимые преобразования, оказывают воздействия на целевую систему и собирают реакции целевой системы.

3.4. CTesK

CTesK – это реализация UniTesK для языка C. CTesK поддерживает разработку спецификаций на спецификационном расширении языка C – SeC (Specification extension of C language). SeC – это ANSI C, к которому был добавлен ряд конструкций, характерных для академических языков формальных спецификаций. В частности, в SeC реализованы пред- и постусловия, инварианты типов данных и глобальных переменных, описатели доступа (access descriptors). Из спецификаций на языке SeC генерируется код на языке C, который затем компилируется обыкновенным компилятором C (в нашем случае MS VC 6.0).

В CTesK реализована архитектура тестового набора UniTesK – тестовые сценарии, оракулы, медиаторы. Реализована поддержка тестирования систем с отложенными реакциями.

4. Применение UniTesK к тестированию встроенного ПО

Применение UniTesK для тестирования встроенного ПО имеет много общего с применением UniTesK для тестирования сетевых протоколов:

- Стимулы и отложенные реакции: в протоколах имеются входящие и исходящие пакеты, во встроенном ПО – процедурные вызовы и прерывания/события;

- Асинхронные операции: пакеты в реализации протокола обрабатываются, как правило, асинхронно; во встроенных системах большинство прерываний происходит также асинхронно;
- Разделение операций на пары "запрос"/"ответ";
- Непроцедурные стимулы и реакции;
- Наличие "слоёв" в реализации, разделение объектов тестирования по "уровням";
- Необходимость в организации удалённого тестирования.

Помимо сходства имеются определённые различия:

- Непроцедурные стимулы встроенного ПО определяются прерываниями аппаратуры, поэтому непроцедурные тестовые воздействия сложнее организовать, чем при тестировании протоколов;
- Непроцедурные реакции связаны с воздействиями на аппаратуру; такие реакции сложнее зарегистрировать, чем реакции протокольных стеков;
- Более тесная связь между асинхронными стимулами: в реализациях протоколов пакеты обрабатываются практически независимо, нехарактерны ситуации, когда параллельная обработка двух и более пакетов влияет на результат обработки отдельного пакета; для встроенного ПО, напротив, асинхронность играет значительную роль, и эта особенность существенно влияет на разработку спецификаций и тестов;
- Встроенное ПО работает в условиях очень ограниченных ресурсов, поэтому размещение тестовой системы на одном устройстве с объектом тестирования невозможно; коммуникационные возможности устройств, как правило, невелики, поэтому организация транспорта стимулов и реакций может оказаться нетривиальной задачей;
- Поведение объекта тестирования в симуляторе и на реальном устройстве могут различаться; это необходимо учитывать при анализе результатов тестирования в симуляторе.

Исследование применимости методологии UniTesK к тестированию встроенного программного обеспечения проводилось на примере TinyOS — системы для разработки встроенного программного обеспечения для беспроводных сетей сенсоров. Исследование показало, что UniTesK пригоден для функционального тестирования встроенного программного обеспечения.

4.1. Функциональное тестирование компонентов TinyOS

С тестированием компонентов можно связать несколько задач:

- Тестирование обычного программного модуля;
- Тестирование программного модуля, связанного с аппаратурой;
- Тестирование конфигурации.

Учитывая особенности программирования в TinyOS, можно выделить следующие задачи:

- Тестирование синхронного кода;

- Тестирование асинхронного кода;
- Тестирование тонкой прослойки над аппаратурой;
- Тестирование расщеплённых операций.

Тестирование программного модуля на языке nesC имеет много общего с тестированием прикладных интерфейсов на других языках, например, C или Java. Основное отличие заключается в тестировании асинхронного кода.

4.1.1. Разработка формальной спецификации

Формальная спецификация разрабатывается средствами CTesK. В CTesK имеются два вида спецификационных функций: стимулы и реакции. В языке nesC есть два вида интерфейсных функций — команды и события. Интерфейсные функции nesC могут быть как стимулами, так и реакциями. Действительно, команда в nesC является стимулом для поставщика интерфейса и реакцией для пользователя интерфейса. Событие в nesC является реакцией для поставщика интерфейса и стимулом для пользователя интерфейса.

Следующее отличие SeC от nesC заключается в том, что реакции в SeC однонаправлены, соответствуют передаче данных от источника получателю. Интерфейсные функции могут иметь возвращаемые значения и обновляемые параметры (updates parameters), то есть представляют двунаправленный обмен данными. Такая ситуация проиллюстрирована на рисунке 4.

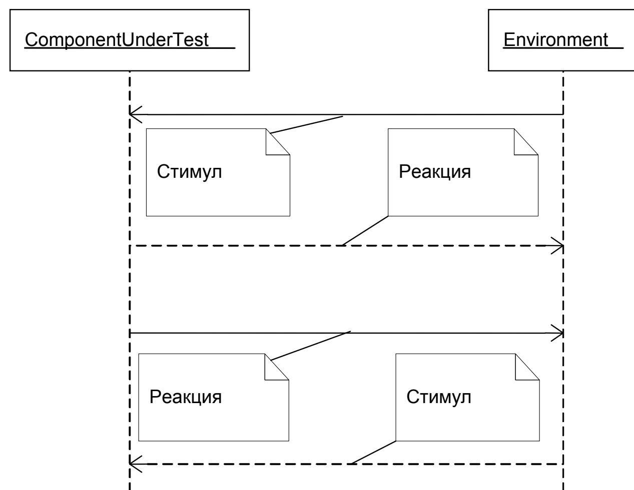

Рис. 4. Пара реакция/стимул.

Ещё одна особенность, которая отличает тестирование компонентов TinyOS от традиционных областей применения UniTesK, заключается во взаимодействии по нескольким интерфейсам. В частности, для выполнения операции на одном интерфейсе, компонент может произвести операции на одном или нескольких других интерфейсах. На рисунке 5 представлен пример: опрос значения

некоторого датчика через промежуточный менеджер атрибутов, который абстрагирует детали обращения к конкретным атрибутам. Результат, возвращаемый в ответ на запрос, зависит от результатов "переговоров" менеджера атрибутов и собственно атрибута.

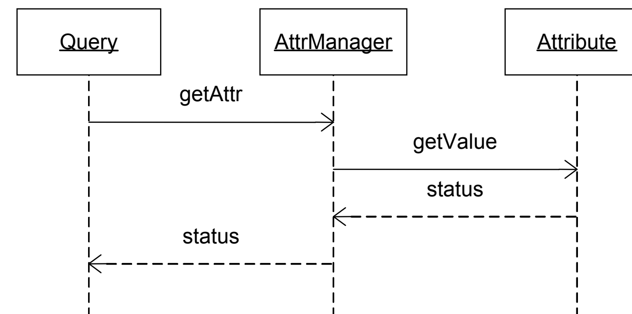

Рис. 5. Взаимодействие на нескольких интерфейсах

Специфика применения UniTesK к TinyOS связана и с асинхронным кодом. Вызов некоторой интерфейсной функции может быть прерван асинхронным вызовом, причём второй вызов может повлиять на результат первого вызова.

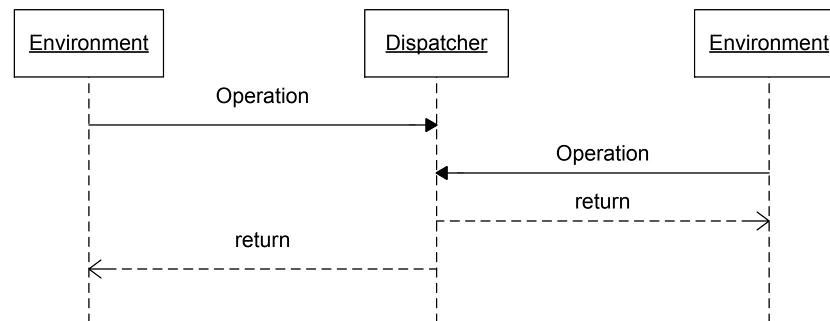

Рис. 6. Асинхронность в TinyOS

Для того, чтобы адекватно промоделировать средствами UniTesK интерфейсные функции компонентов nesC, мы предлагаем разделить вызовы команд и событий nesC на две составляющие: вызов и возврат управления. При этом поток данных от окружения к целевому компоненту специфицируется как стимул, а обратный поток данных – как реакция. Данное решение позволяет единообразно моделировать расщеплённые операции и операции, которые возвращают управление немедленно.

Большинство компонентов TinyOS не связано с аппаратурой, поэтому все стимулы и реакции таких компонентов состоят из вызовов интерфейсных методов. По другому обстоят дела с компонентами, которые непосредственно взаимодействуют с аппаратурой. Для таких компонентов имеются непроцедурные стимулы — прерывания аппаратуры, и непроцедурные реакции — операции с "железом".

С точки зрения разработки формальных спецификаций компоненты, взаимодействующие с аппаратурой, ничем особенным не отличаются. Непроцедурные стимулы можно моделировать спецификационными функциями-стимулами, а для описания воздействия на аппаратуру есть две стратегии:

- Ввести спецификационные функции-реакции, которые соответствуют обращению программного обеспечения к определённым аппаратным функциям;
- Включить модель состояния аппаратуры и представить изменения состояния аппаратуры в постусловии.

Первая стратегия применима в тех ситуациях, когда возможно перехватывать обращения к аппаратуре. Вторая стратегия является уместной, когда есть возможность наблюдать изменения состояния аппаратуры.

Конфигурации собираются из нескольких компонентов — модулей и более мелких конфигураций. Конфигурации не содержат собственного кода, все внешние интерфейсы конфигурации реализуются в компонентах, составляющих конфигурацию. Стимулы и реакции конфигурации составляются из стимулов и реакций внешних интерфейсов конфигурации. Если в состав конфигурации входят компоненты, взаимодействующие с аппаратурой, то к стимулам и реакциям конфигурации относятся также и операции с аппаратурой. С точки зрения методики разработка спецификаций для модулей и конфигураций не различаются.

4.1.2. Тестовые сценарии для компонентов

Особенности разработки тестовых сценариев UniTesK для TinyOS вытекают из уже упоминавшихся особенностей TinyOS и nesC:

- Разделение исполнения на синхронное и асинхронное;
- Взаимодействие по нескольким интерфейсам;
- Наличие расщеплённых операций;
- Наличие взаимодействия с аппаратурой.

Если целевой компонент не вызывает другие компоненты для обработки запроса, то разработка тестового сценария или сценарного метода для такой операции ничем не отличается от тестирования обычного метода в UniTesK. Если целевой компонент для выполнения запроса обращается к окружению, то для тестирования необходимо подготовить соответствующие заглушки, имитирующие ответы от окружения. Задача заглушек — передавать в

тестируемый компонент различные значения изменяемых параметров и возвращаемых значений.

Поскольку для тестирования представляет интерес проверка поведения целевого компонента при различных ответах окружения, в тестовом сценарии необходимо уметь перебирать ответы окружения. Наиболее естественно для этой задачи подходит использование в сценарных методах UniTesK итераторов и “stable”-переменных. На рисунке 7 схематично изображён сценарный метод, реализующий предложенный приём.

Методы разработки тестовых сценариев для компонентов, взаимодействующих с аппаратурой, не отличаются от уже рассмотренных приёмов. Тестовый сцена-рий перебирает параметры запроса и значения, которые должна получить реа-лизация от аппаратуры. Вопрос доставки значений аппаратуры выходит за рам-ки разработки тестового сценария, и будет рассмотрен в разделе про медиаторы.

```
bool scenario test_with_stubs() {
iterate( /* перебор аргументов запроса */ ){
iterate( /* перебор ответов окружения */ ) {
/* передаём тестовому агенту ответы окружения */
setup_env( ... );
/* Вызываем целевую операцию */
call_target_operation( /* аргументы */ );
}
}
return true;
```

Рис. 7. Пример перебора параметров запроса и ответов окружения.

4.1.3. Тестирование асинхронного кода

При тестировании асинхронного кода необходимо создавать тестовые ситуации, в которых выполнение целевой функции прерывается, причём для систематического тестирования необходимо, чтобы можно было управлять тем, в каком месте исполнение будет прервано. Для реализации управляемого прерывания исполнения целевой процедуры мы предлагаем следующее:

1. Инструментировать код целевого метода: при инструментировании в код внедряются заглушки; когда исполнение доходит до заглушки, исполнение целевой функции может быть прервано.
2. В тестовом сценарии задаётся, в каких заглушках целевая функция будет прервана при прогоне теста, и какие функции прервут исполнение.

Во время написания статьи в симуляторе TinyOS прерывание исполнения не поддерживалось. Предложенный подход к симуляции асинхронного исполнения функций может позволить протестировать поведение функции при прерываниях даже в условиях фактически непрерываемого исполнения.

Как правило, асинхронное поведение в TinyOS наблюдается в функциях, которые вызываются из обработчиков прерываний. Для достоверного тестирования асинхронного поведения на аппаратуре необходимо удалить источники прерываний, в цепочке обработки которых встречается целевая функция. Вместо аппаратных прерываний необходимо организовать программные прерывания, что можно сделать аналогично симулятору — через инструментирование кода. В этом случае тестовый сценарий определяет, в каких точках исполнения целевой функции необходимо прерывать исполнение, и настраивает заглушки.

Оценка затрат ресурсов на тестирование асинхронного поведения при помощи инструментирования кода:

- Дополнительный код: *число вставок * размер вставки + размер диспетчера* (здесь диспетчер означает компонент, который эмулирует асинхронное поведение); размер вставки — несколько инструкций, размер диспетчера до 1 Кб;
- Расходы на данные: *число вставок * размер данных вставки*; размер данных для отдельной вставки можно оценить как несколько байт на отдельную вставку.

Как показывают оценки, предложенный подход к тестированию асинхронного поведения возможно реализовать на типовых устройствах для TinyOS.

Предложенный подход к тестированию асинхронного выполнения ПО пока не применялся на практике.

4.1.4. Медиаторы для тестирования компонентов TinyOS

Размеры характерных устройств не позволяют разместить на этих устройствах полученный тестовый набор. По этой причине необходимо проводить удалённое тестирование целевых компонентов.

4.1.4.1. Тестирование компонентов, не взаимодействующих с аппаратурой

Необходимо окружить целевой компонент заглушками. Назначение заглушек — передавать тестовые воздействия на целевой компонент и регистрировать реакции целевого компонента. Связь с заглушками осуществляется с использованием некоторого транспортного механизма. Демультиплексирование данных осуществляется специальным компонентом, который мы назвали Messenger.

Было реализовано удалённое тестирование синхронного кода в симуляторе TinyOS. В качестве транспортного механизма использовался протокол TCP/IP. Для удалённого тестирования на устройстве необходимо основывать транспортный механизм на тех коммуникационных средствах, которые предоставляет устройство. Типовые устройства для TinyOS содержат параллельный порт (COM-порт), а в поставку TinyOS входят компоненты, реализующие приём и передачу данных по параллельному порту.

Мы оцениваем затраты на транспорт между устройством и тестовой системой величинами около 2-3 Кб кода и 1 Кб данных на устройстве. Эта оценка основана на оценке ресурсов реализации компонента Messenger для симулятора. Затраты стека для передачи и приёма данных можно оценить примерно в 10 вызовов. Эти оценки показывают, что UniTesK применим для тестирования компонентов на устройстве. При выполнении проекта по исследованию применимости UniTesK к тестированию TinyOS тестирование на реальных устройствах не проводилось.

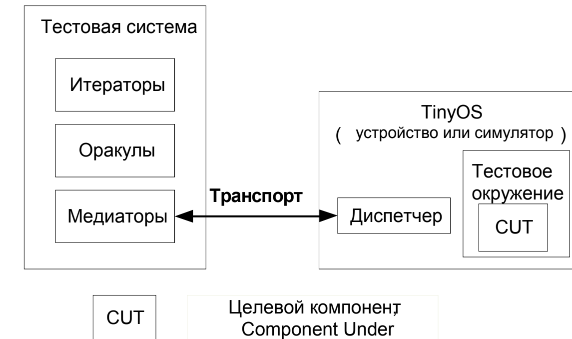

Рис. 8. Удалённое тестирование компонентов TinyOS.

4.1.4.2. Тестирование взаимодействия с аппаратурой

Такое тестирование средствами UniTesK требует разработки специализированных медиаторов, которые позволяют решить следующие задачи:

- Подача тестовых воздействий через аппаратуру или симулятор аппаратуры;
- Регистрация операций с аппаратурой;
- Определение состояния аппаратуры.

Остальные компоненты тестового набора могут разрабатываться в рамках методологии UniTesK без особых изменений.

Тестирование компонентов, связанных с аппаратурой, не проводилось.

5. Результаты применения UniTesK к тестированию компонентов TinyOS

В 2003-2004 годах совместно с компанией Luxoft проводился проект по исследованию применимости UniTesK к тестированию компонентов TinyOS и сетей устройств под управлением TinyOS.

В проекте преследовались следующие цели:

1) Разработать методику применения UniTesK к тестированию приложений TinyOS;
2) Показать применимость подхода UniTesK к верификации сенсорных сетей.

В качестве объекта тестирования был выбран компонент Attr из подсистемы управления атрибутами в TinyDB. Мы провели тестирование реализации шести методов, принадлежащих двум интерфейсам компонента. Функциональные требования извлекались из документации и исходных текстов. Из-за скудости документации нам пришлось использовать исходные тексты реализации. Всего было извлечено порядка 40 функциональных требований.

5.1. Использованный процесс

Метод: набор инструментов CTesK. Данный проект отличался от прочих проектов с использованием UniTesK тем, что (1) среди методов интерфейса имеются асинхронные; (2) обработка операции на одном интерфейсе может привести к исполнению операции на другом интерфейсе.

Разработка: Разработка проводилась в среде Linux с использованием набора инструментов CTesK, средств разработки TinyOS и обычных средств разработки Unix (Emacs, make).

Устройство тестового стенда и средства тестирования: тестируемый компонент был развёрнут в симулятор TinyOS для Linux-PC.

Тестирование проводилось в удалённом режиме. Были разработаны специализированные компоненты TinyOS, обеспечивающие связь между тестовой системой и тестируемым компонентом.

5.2. Затраты

Продолжительность проекта: 2 месяца.

Трудозатраты: 2 человеко-месяца.

5.3. Результаты проекта

Протестированы 6 интерфейсных методов. Размеры компонентов тестового набора:

Компонент тестового набора	**Размер, в строках кода**
Спецификации	550
Тестовые сценарии	200
Медиаторы	500

Таб. 1. Компоненты тестовой системы.

Общее число тестовых сценариев: 5. Общее число тестовых случаев: порядка 50. Число обнаруженных ошибок: 1.

При выполнении проекта удалось достичь поставленных целей. Была показана применимость UniTesK к тестированию встроенных систем. Были разработаны методы применения UniTesK к тестированию встроенных систем.

6. Применимость UniTesK для тестирования встроенных приложений

Проект показал, что метод UniTesK применим для тестирования встроенных приложений.

Сильные стороны UniTesK в контексте тестирования встроенного программного обеспечения состоят в следующем:

- Формализация требований к ПО на языке, близком языку разработки; это упрощает анализ требований и может ускорить разработку системы;
- Гибкие средства описания недетерминизма в формальной модели;
- Относительно простые средства описания асинхронных операций;
- Удобные средства для описания взаимодействия с аппаратурой — операций над состоянием аппаратуры и аппаратных прерываний;
- Гибкие средства размещения тестового стенда.

Слабые стороны UniTesK в контексте тестирования встроенного ПО:

- Сложность организации транспорта тестовых воздействий (с другой стороны, эти трудности должны быть присущи всем средствам тестирования встроенного ПО);
- Сложность тестирования асинхронных операций (но это в принципе не может быть простой задачей);
- В случае асинхронных операций требуется много рутинной работы для манипулирования состоянием модели; ряд таких операций можно было бы автоматизировать, если ввести в язык средства, характерные для темпоральных логик (но этот вопрос выходит за рамки данной работы);
- Сложность формальных спецификаций UniTesK для статического анализа встроенного программного обеспечения (аналитическая верификация, *model-checking* и т.п.).

6.1. Спецификации UniTesK и статический анализ

Распространённый подход к верификации встроенного программного обеспечения основан на статическом анализе системы. К статическому анализу мы относим процедуры исследования свойств ПО по априорным описаниям или моделям. Примерами статического анализа систем могут служить аналитическая верификация и *model-checking*.

В методах статического анализа систем широко используются формальные спецификации для исследования свойств программных систем (таких, как отсутствие тупиковых ситуаций, обязательное завершение исполнения, временные характеристики и т.п.).

Интересно сопоставить формальные спецификации UniTesK и формальные спецификации для статических методов. Можно задать следующие вопросы:

- Применимы ли формальные спецификаций UniTesK для проведения априорных рассуждений о свойствах описываемых программ?
- Применимы ли формальные спецификации, которые используются в такого рода исследованиях, для тестирования?

Как показывает практика, спецификации для анализа и спецификации для тестирования существенно различаются по своим свойствам. Спецификации для тестирования предоставляют разработчику гибкие, но трудно верифицируемые средства. Спецификации для анализа требуют применения строго ограниченных средств, которые стесняют разработчика спецификаций или трудно применимы для автоматического построения оракула.

К примеру, в спецификациях CTesK разрешается использование указателей, циклов и произвольных выражений языка C. Как нам представляется, наши спецификации можно (в принципе) использовать для анализа и доказательства теорем о свойствах описываемой системы, но для этого надо, как минимум, запретить указатели, циклы с переменными границами (ограничители времени исполнения) и ограничить выражения, которые могут встречаться в пред- и постусловиях. Мы считаем, что при выполнении указанных требований язык CTesK станет в один ряд с такими языками, как VDM, RaiseSL, а для них имеются системы аналитической верификации. Правда, при этом пропадёт основное преимущество языка CTesK — приближенность к конечному пользователю.

На применимость методов статического анализа влияет ещё один аспект — сложность модели(спецификации). Как показывает наша практика, модель, достаточно детальная для построения оракула, оказывается слишком сложной для автоматического анализа. Анализ возможен только для относительно простых моделей, которые могут дать лишь общий оракул, недостаточный для целей тестирования.

Вывод: спецификации, которые используются в UniTesK, могут, в принципе, использоваться для статического анализа встроенного ПО. На практике этому препятствует гибкость выразительных средств, которые используют разработчики спецификаций, и сложность моделей, необходимых для построения оракулов.

7. Заключение

UniTesK удобен для функционального тестирования встроенного ПО:

- Формальные спецификации UniTesK удобны для формализации требований к встроенному программному обеспечению;
- Тестовые сценарии позволяют компактно записывать сложные тестовые последовательности;

- Механизм медиаторов предоставляет гибкие средства для отделения сценариев и формальной модели от реализации, позволяет строить различные схемы развёртывания тестового стенда.

Спецификации, которые используются в UniTesK, могут, в принципе, использоваться для статического анализа встроенного ПО. На практике этому препятствует гибкость выразительных средств, которые используют разработчики спецификаций, и сложность моделей, необходимых для построения оракулов.

Литература

1. J. Hill. A Software Architecture Supporting Networked Sensors. Masters thesis, December 2000.
2. I. Bourdonov, A. Kossatchev, V. Kuliamin, A. Petrenko. UniTesK Test Suite Architecture. Proceedings of FME 2002. LNCS 2391, pp. 77-88, Springer-Verlag, 2002.
3. Веб-сайт CTesK. http://www.unitesk.com/products/ctesk/
4. D. Gay, P. Levis, R. von Behren, Matt Welsh, E. Brewer, D. Culler. The nesC Language: A Holistic Approach to Networked Embedded Systems. Proceedings of Programming Language Design and Implementation (PLDI) 2003, June 2003.
5. D. Gay, P. Levis, David Culler, E. Brewer. nesC 1.1 Language Reference Manual. http://nescc.sourceforge.net/papers/nesc-ref.pdf
6. S. Madden, J. Hellerstein, W. Hong. TinyDB: In-Network Query Processing in TinyOS. http:// ftp://download.intel.com/research/library/IR-TR-2002-47-120520020948_109.pdf
7. I. Bourdonov, A. Kossatchev, A. Petrenko, D. Galter. KVEST: Automated Generation of Test Suites from Formal Specifications. FM’99: Formal Methods. LNCS, volume 1708, Springer-Verlag, 1999, pp. 608–621.
8. И. Б. Бурдонов, А. В. Демаков, А. С. Косачев, А. В. Максимов, А. К. Петренко. Формальные спецификации в технологиях обратной инженерии и верификации программ. Труды Института системного программирования, 1999 г., том 1, стр. 35-47.