LDRA
Software Technology

экспонента
Центр Инженерных Технологий
и Моделирования

Развитие верификации объектного кода

Билл Сент-Клер из LDRA рассматривает растущую потребность в проверке объектного кода при разработке программного обеспечения

Статья Билла Сент-Клер изначально была опубликована в журнале
Automotive Electronics Feb/Mar 2006

Развитие верификации объектного кода

Постоянно растущая зависимость от программного обеспечения означает, что многие компании из автомобильного и других секторов, которые не имеют традиционного требования к сложному анализу программного обеспечения, теперь вынуждены проводить тесты безопасности и надежности, связанные с характером приложений, которые находятся в разработке.

Благодаря этому повышенному требованию к тестированию программного обеспечения в различных отраслях, появилась тенденция к тому, что компании выходят за пределы своего собственного рыночного сектора для поиска наилучших методов или стандартов. Примеры такого отраслевого пересечения были замечены в автомобильной и авиационных отраслях промышленности с принятием элементов стандарта DO-178B в первом случае и аналогичным внедрением стандарта MISRA в последнем.

Из-за применения стандартов тестирования из других секторов существует потенциал для незнакомых методов тестирования. Это иллюстрируется, среди прочего, требованиями к верификации объектного кода стандарта DO-178B. Являясь ключевым элементом тестирования многих программ для авионики, она была относительно неиспользуемой техникой вне этой отрасли.

Возрастающая сложность и критический характер безопасности многих современных встроенных приложений управления означает, что, когда поставщики, не входящие в состав авиапромышленности, принимают DO-178B, верификация объектного кода является одним из ключевых элементов, который им приходится принять к сведению.

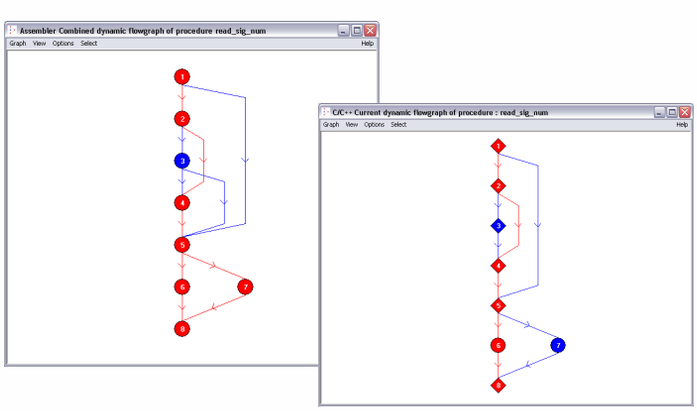

Рисунок 1. Примеры графа вызова для кода на языках высокого и низкого уровня

Верификация объектного кода

Что же такое верификация объектного кода?

Раздел стандарта DO-178B (КТ-178B) 6.4.4.2 «Анализ структурного покрытия» описывает это требование как:

«Анализ структурного покрытия может быть выполнен по «Исходному коду», за исключением, когда: уровень ПО есть А и компилятор генерирует объектный код, который не трассируется прямо на операторы «Исходного кода».

В этом случае следует провести дополнительную верификацию объектного кода, чтобы убедиться в правильности таких сгенерированных последовательностей. Примером объектного кода, который не прямо трассируется в «Исходный код», является генерируемая компилятором проверка границ массива.»

Итак, верификация объектного кода связана с тем, насколько структура потока управления, сгенерированного компилятором объектного кода отличается от структуры исходного кода приложения, из которого он был получен. Такие различия могут возникать по ряду причин, например, интерпретация компилятора, оптимизация и т.д. Учитывая, однако, что традиционные методы структурного покрытия применяются на уровне исходного кода, тогда как на самом деле это объектный код, который выполняется на процессоре, различия в структуре потоков управления между ними могут вызвать значительные пробелы в процессе тестирования.

Требования DO-178B таковы, что разработчики приложений, которые подчиняются стандарту, должны реализовать средства верификации объектного кода для элементов приложения, которые имеют уровень безопасности А.

Хотя такие элементы часто являются компонентами приложения в целом, их верификация может тем не менее представлять собой значительную часть усилий по тестированию и, следовательно, требует значительных ресурсов с точки зрения времени и денег.

Таким образом, возможность внедрения автоматизированных процессов, основанных на компиляторах, может помочь значительно снизить общие затраты на разработку.

Решения для верификации объектного кода

Рынок разработки программного обеспечения признал и отреагировал на растущие требования к средствам верификации объектного кода, исходящих из разных отраслей промышленности, и многие поставщики программного обеспечения теперь могут предоставлять либо частичные, либо полные решения для анализа структурного покрытия исходного и объектного кода от компонентных до системных и интеграционных уровней.

Различные решения на рынке, как правило, используют комбинации вариантов инструментов для исходного кода высокого уровня и уровня ассемблера, определяемым целевым процессором, для которого требуется выполнение приложения. Типичным примером может быть комбинация C/C ++ как языка высокого уровня и ассемблера TMS320C25x на уровне объектного кода с версиями соответствующих инструментов, объединенных вместе для обеспечения необходимых средств сбора структурного покрытия. Многие другие сочетания языков высокого уровня и ассемблера поддерживаются различными поставщиками инструментов и примеры хорошо известных показателей покрытия, которые обычно поддерживаются этими решениями, перечислены ниже.

- Покрытие операторов
- Покрытие ветвления
- Покрытие вызова процедур и функций
- Покрытие булевых выражений
- Покрытие ветвей и решений
- Покрытие комбинации ветвей и условий
- Модифицированное покрытие условий / решений (DO-178B) *
  (* Зависит от языка)

Решения для верификации объектного кода

Некоторые поставщики инструментов сделали значительный шаг вперед, расширив свои решения по проверке объектного кода, чтобы обеспечить частичные или полностью автоматизированные средства, которые ориентированы на уровень модульного тестирования и, следовательно, позволяют применять этот сложный метод анализа на гораздо более ранней стадии жизненного цикла разработки ПО.

Этот режим Object Box, как его называют некоторые поставщики инструментов для модульного тестирования объектного кода, позволяет пользователям создавать тестовые вектора для структурного покрытия исходного кода высокого уровня и применять эти те же самые тестовые вектора к структурному покрытию соответствующего объектного кода.

Ключом к этому решению является создание расширенной программы драйверов, которая в зависимости от сложности решения поставщика автоматически создается или создается с помощью ручных или частично автоматизированных средств. Этот драйвер инкапсулирует всю тестовую среду, определяя, запуская и контролируя тестовые вектора посредством начальной проверки теста, а затем последующего регрессионного анализа. В режиме Object Box этот драйвер может быть связан либо с блоком исходного кода высокого уровня или ассоциированным объектным кодом. При этом пользователи могут гарантировать, что единые будут применяться и сравниваться одни и те же тестовые вектора, что выявит любые несоответствия / недостатки.

Если выявлены расхождения / недостатки структурного покрытия, пользователям затем предоставляется возможность определить дополнительные тестовые вектора, чтобы закрыть любые пробелы в процессе тестирования. Очевидным преимуществом того, что можно определить и применить корректирующие действия на такой ранней стадии разработки программного обеспечения, является то, что это намного проще и дешевле. Это также значительно повышает качество кода и общего процесса тестирования, при этом последний получает преимущества на более поздних этапах интеграции и тестирования системы, а затем и в виде уменьшенных показателей отказов и расходов на обслуживание, когда приложение уже было развернуто в рабочей среде.

Несмотря на то, что код все еще находится в разработке, вместе с удовлетворением критериев объектного кода с помощью высокоавтоматизированного и экономически эффективного способа, разработчики также могут воспользоваться значительной дополнительной обратной связью, которая предоставляется инструментами тестирования в виде рассмотрений

исходного кода и дизайна.Результаты этих средств анализа могут быть возвращены команде разработчиков с возможностью выявления и устранения недостатков кода и дизайна, что еще больше повысит качество приложения в целом.

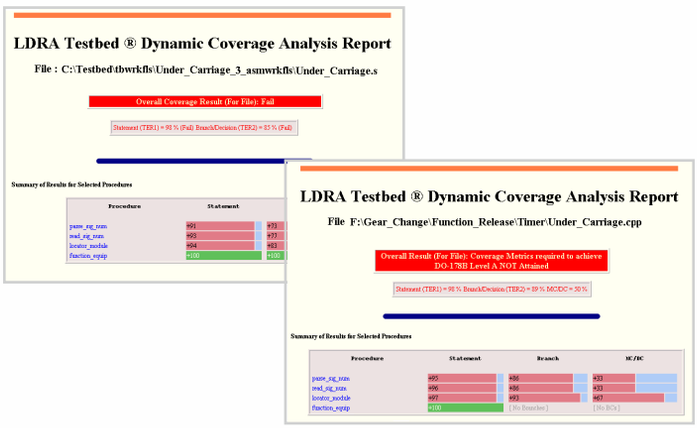

Рисунок 2. Примеры отчета по динамическому анализу высоко- и низкоуровнего кода

Вывод

Нет сомнений в том, что верификация объектного кода представляет собой серьезную проблему для тех проектов разработки программного обеспечения, в которых необходимо ее проведение. Однако при наличии правильных инструментов и средств масштаб этих проблем может быть значительно сокращен, что позволит разработчикам реализовать весь потенциал и преимущества, которые предоставляет такой анализ, путем повышения качества и надежности кода.

Билл Сент-Клер — технический евангелист в LDRA