Проведение итеративного динамического анализа приложений, предоставляющих графический интерфейс пользователя[1]

М. К. Ермаков <mermakov@ispras.ru>
А. Ю. Герасимов <agerasimov@ispras.ru>
Д. О. Куц <kutz@ispras.ru>
А. А. Новиков <a.novikov@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В настоящее время в промышленной разработке программного обеспечения с графическим пользовательским интерфейсом преобладают полуавтоматические подходы к тестированию, требующие участия эксперта для создания наборов тестовых сценариев. Повышение сложности программных систем приводит к снижению эффективности применения методов, полагающихся на участие эксперта в процессе тестирования. С учётом возрастания доступности и снижения стоимости вычислительных ресурсов становятся экономически выгодны методы автоматического анализа программ. В рамках данной статьи предлагается метод полностью автоматического динамического анализа программ, предоставляющих графический пользовательский интерфейс. Среди существующих инструментов тестирования и анализа программного обеспечения на основе обзора, приведённого в статье, выделяется свободно распространяемое программное инструментальное средство GUITAR, обеспечивающее максимальную степень автоматизации. Рассматриваются основные ограничения подхода, реализованного в средстве GUITAR: недостаточная степень точности модели графического интерфейса, недостаточность полноты описания атрибутов элементов графического интерфейса. Данные ограничения приводят к невозможности покрытия наборами тестовых воздействий отдельных элементов графического интерфейса в рамках анализа и созданию тестовых сценариев, которые не могут быть воспроизведены на практике. В статье предлагается ряд модификаций подхода: итеративное построение модели графического интерфейса, расширение списка атрибутов элементов графического интерфейса, алгоритм итеративного построения тестовых наборов по графу потока событий с целью явной проверки функциональности всех элементов графического интерфейса. В статье рассмотрены результаты практических экспериментов применения предложенных модификаций для набора проектов с открытым исходным кодом, демонстрирующие

[1] Работа проводится в рамках научно-исследовательских работ Института системного программирования РАН в 2014 – 2017 годах

повышение эффективности анализа и полноты покрытия графического интерфейса создаваемыми тестовыми наборами. В заключение статьи обсуждаются перспективные направления дальнейшей работы, включающие применение методов символьного исполнения и анализа помеченных данных.

Ключевые слова: динамический анализ программ; анализ программ; тестирование GUI, тестовое покрытие.

DOI: 10.15514/ISPRAS-2017-29(1)-8

Для цитирования: Ермаков М.К., Герасимов А.Ю., Куц Д.О., Новиков А.А. Проведение итеративного динамического анализа приложений, предоставляющих графический интерфейс пользователя. Труды ИСП РАН, том 29, вып. 1, 2017 г., стр. 119-134. DOI: 10.15514/ISPRAS-2017-29(1)-8

1. Введение

В повседневной жизни современному человеку всё чаще приходится сталкиваться с использованием высокотехнологичных устройств как для выполнения работы, так и для общения, отдыха и развлечений. Ключевым элементом, обеспечивающим разнообразие функциональных возможностей высокотехнологичных устройств, является программное обеспечение, под управлением которого работает устройство. Во избежание экономического и материального ущерба, а также для обеспечения безопасности человека при использовании устройств, предъявляются высокие требования к качеству программного обеспечения. Стоимость исправления ошибки в программном обеспечении значительно возрастает при приближении к концу жизненного цикла. В связи с этим разработка средств обнаружения ошибок на ранних этапах разработки программ является крайне актуальной задачей.

Анализ программного кода на наличие ошибок и уязвимостей может производиться как вручную, что требует большого количества времени и трудозатрат, так и полуавтоматически и автоматически. Наиболее перспективным подходом является применение методов и технологий, позволяющих частично или полностью автоматизировать отдельные этапы процесса анализа программ. Среди данных подходов можно выделить группу методов динамического анализа, основывающихся на исследовании программного обеспечения во время его выполнения. Данная группа методов нацелена на построение и проверку сценариев использования программного обеспечения, близких к ожидаемым от конечных пользователей. Одной из основных проблем подобных методов является необходимость обеспечить достаточную полноту покрытия кода программы минимальным набором сценариев работы с программой. В рамках задачи автоматизации проверки программ можно выделить полуавтоматические методы, основанные на участии эксперта в создании базы сценариев для проверки, и полностью автоматические методы, ориентированные на извлечение информации о

структуре приложения с целью генерации тестовых сценариев без участия эксперта.

В то же время, в связи с бурным развитием устройств с возможностью предоставления графического пользовательского интерфейса, особый практический интерес при разработке программного обеспечения представляет анализ поведения именно таких программ. Ошибки проектирования и нарушение корректности работы обработчиков событий от элементов графического пользовательского интерфейса могут приводить к невозможности использования целевой функциональности программ, а тестирование и анализ непосредственно самого графического интерфейса становятся необходимым этапом при создании программных продуктов.

Для проверки качества реализации программ с графическим пользовательским интерфейсом традиционно используются инструменты автоматизации анализа и тестирования, которые ограничиваются тем, что исключают необходимость вручную взаимодействовать с элементами графического интерфейса. Как будет показано в обзоре области, представленном в следующей главе, данные инструменты фокусируются на точности определения соответствия между элементами, указанными в сценарии тестирования, и реальными элементами, отображаемыми в процессе работы целевой программы.

Данная работа нацелена на развитие методов полностью автоматического динамического анализа программ, предоставляющих графический интерфейс пользователя, позволяющих минимизировать степень участия эксперта в процессе создания набора тестовых сценариев, сохраняя при этом полноту покрытия программы.

Далее статья имеет следующую структуру — в разд. 2 представлен обзор существующих инструментов полуавтоматического и автоматического анализа программ, предоставляющих графический пользовательский интерфейс. Разд. 3 описывает общие принципы работы одного из наиболее перспективных инструментов тестирования программ с графическим пользовательским интерфейсом (GUITAR), выбранного в рамках данной работы в качестве основы для реализации. Разд. 4 описывает модификации, внесенные в инструмент GUITAR в рамках данной работы. Разд. 5 посвящен описанию эксперимента и оценке практических результатов применения модифицированной версии инструмента GUITAR для ряда свободно распространяемых приложений. В заключении приводится общая оценка проделанной работы и рассматриваются наиболее перспективные направления дальнейших исследований.

2. Обзор существующих решений для анализа программ с графическим пользовательским интерфейсом

На сегодняшний день существует множество инструментов, осуществляющих анализ приложений с графическим интерфейсом. Большинство из них основано

на полуавтоматическом подходе, который заключается в записи действий пользователя с приложением и последующем их воспроизведении. Инструмент производит запуск исследуемого приложения, регистрирует элементы графического интерфейса и взаимодействие между ними, в то время как пользователь производит некоторые действия с приложением, такие как нажатие кнопок, ввод данных в текстовые поля и пр. На основе собранной информации о структуре графического интерфейса приложения пользователю предлагается возможность составить последовательности действий над выделенными графическими элементами, а также программный интерфейс, с помощью которого можно описывать более сложные тестовые сценарии.

Также некоторые современные средства предлагают возможности, в частности для Java-приложений, модульного тестирования (unit testing) для проверки корректности сценариев работы графического приложения. Очевидно, что такой подход также позволяет сократить затраты на построение тестового покрытия – сценарии работы отдельного графического элемента можно непосредственно использовать при составлении сценария работы приложения.

В качестве примеров средств, предлагающих пользователю описанные выше возможности проведения тестирования, можно привести следующие инструменты.

Ranorex [1] – проприетарный инструмент автоматизации тестирования приложений, разработанных под операционную систему Windows. Ranorex позволяет работать с приложениями, основанными на распространенных для Windows библиотеках графического интерфейса, предоставляет возможности тестирования Java-приложений, основанных на библиотеке SWT. Также Ranorex поддерживает тестирование приложений на мобильных платформах, таких как Android и iOS. Инструмент представляет графический пользовательский интерфейс программы в виде дерева (леса), где корнем является главное окно (или несколько окон) программы, промежуточными узлами являются контейнеры элементов графического пользовательского интерфейса, а листьями — конечные элементы, такие как поля ввода, экранные кнопки и др., и использует XPath-подобный язык RanoreXpath для описания последовательностей взаимодействия с элементами GUI, а также предоставляет пользователям возможность описывать в виде программы базовые взаимодействия с приложением на языках C#, VB.net и IronPython.

Abbot [2] – среда автоматизации тестирования для Java-приложений с открытым исходным кодом. Данный инструмент позволяет динамически идентифицировать элементы графического приложения по набору атрибутов. Таким образом исключается привязка элемента к его положению в интерфейсе, что делает анализ менее зависимым от изменения положения элементов графического пользовательского интерфейса.

Maveryx [3] – это инструмент автоматизированного тестирования приложений с графическим интерфейсом, реализованных на языке Java, в том числе и для

платформы Android. Отличительной особенностью этого инструмента является то, что для создания и воспроизведения тестов не требуется модель, описывающая графические элементы. В Maveryx реализован принцип динамического определения структуры графического интерфейса приложения при воспроизведении каждого тестового сценария, что повышает гибкость и эффективность проведения тестирования.

Squish [4] – проприетарный инструмент автоматизации тестирования приложений с графическим интерфейсом для платформ Java AWT/Swing, SWT, Windows, Android и пр. Squish не требует модификации кода тестируемого приложения. Для тестирования приложения используется специальный управляющий модуль, который внедряется в адресное пространство приложения, данный модуль запускает приложение и связывается с инструментом. Squish имеет возможность проверки корректности выполнения тестов, сопоставление состояний некоторых виджетов производится с помощью анализа снимков экрана (screenshots).

Sikuli [5] – среда автоматизации тестирования приложений. Особенность данного инструмента заключается в предоставлении пользователю возможности строить тесты приложения, основываясь на изображениях графических элементов приложения. Для поиска и взаимодействия с элементами графического интерфейса используются их снимки (screenshots).

Инструмент GUITAR [6] выделяется среди всех рассмотренных решений тем, что позволяет проводить полностью автоматическое тестирование графических Java-приложений, основанных на AWT/Swing, SWT, а также приложений для платформы Android. GUITAR производит анализ структуры графического интерфейса на основе исполняемых файлов приложения и составляет модель данной структуры. На основе этой модели автоматически строится множество тестовых наборов – возможные последовательности действий определенной длины.

В то же время, у инструмента GUITAR есть недостатки, которые не позволяют производить анализ программы в полной мере. В главе 3 подробно рассматриваются устройство, особенности и ограничения инструмента GUITAR.

3. Особенности инструмента GUITAR

Инструмент GUITAR состоит из четырех компонентов, схема взаимодействия которых представлена на рис. 1.

Рис. 1. *Схема работы GUITAR*

Fig. 1. *GUITAR tool scheme*

3.1 Алгоритм работы компонента Ripper

Компонент Ripper проводит анализ окон, открываемых приложением, и сохраняет информацию об элементах этих окон в формате XML. В каждом окне присутствует набор графических элементов. Все элементы объединены в общую иерархию — самым верхним уровнем является непосредственно само окно, далее лежат графические контейнеры верхнего уровня (такие как, например, группа вкладок). Данные контейнеры соответствуют базовым элементам библиотек построения графического интерфейса (например, Swing, SWT и др.). Содержимое каждого контейнера состоит из контейнеров более низкого уровня и конкретных графических элементов. В начальный момент времени для анализа доступно лишь главное окно приложения. Во время анализа окон Ripper производит для каждого видимого в данный момент графического элемента (кнопка, поле ввода, выпадающий список и т. д.) действие, соответствующее данному элементу (например, нажатие для кнопки). При проведении действий сохраняется информация об изменении графической структуры (например, открытие нового окна или закрытие текущего). В случае открытия новых окон при выполнении действия с некоторыми элементами проводится анализ данного окна.

3.2 Построение графа потока событий и тестового покрытия

Далее описание графической структуры, построенной компонентом Ripper, переводится в ориентированный граф потока событий (Event Flow Graph), который описывает взаимосвязи между отдельными графическими элементами на основе эффектов событий, соответствующих данным элементам. В частности, наличие ребра, следующего из события A в событие B, означает, что если в некоторый момент времени было совершено действие над элементом графического интерфейса, соответствующее событию A, то графическое приложение перешло в состояние, в котором возможно совершение действия, соответствующего событию B. На основе графа потока событий происходит генерация тестовых сценариев взаимодействия с программой. Каждый тестовый сценарий представляет собой последовательность событий.

Компонент Replayer осуществляет воспроизведение набора тестовых сценариев с целью обнаружения ошибок и проверки корректности работы программы.

3.3 Полнота анализа

Таким образом, полностью автоматически осуществляется генерация тестовых наборов и запуск приложения на этих наборах. Однако, как уже было сказано, использование инструмента GUITAR не позволяет добиться достаточной полноты покрытия. Рассмотрим простой пример, иллюстрирующий вышесказанное.

Предположим, одно из окон приложения имеет два переключателя и кнопку. Нажатие кнопки при активированном первом переключателе открывает диалоговое окно A, нажатие кнопки при активированном втором переключателе открывает диалоговое окно B. Во время анализа компонент Ripper совершает действия над элементами в той последовательности, в которой элементы доступны из данного окна. При этом каждое действие совершается один раз. Допустим, что элементы обрабатываются в следующей последовательности: переключатель №1, переключатель №2, кнопка. Тогда Ripper активирует сначала переключатель №1, затем переключатель №2 (при этом переключатель №1 деактивируется), затем произведет нажатие кнопки. Откроется диалоговое окно B, Ripper произведет анализ нового окна. Таким образом, сведения о существовании окна A и его элементах не будут доступны на последующих этапах анализа.

Данный пример характеризует основную проблему анализа, проводимого инструментом GUITAR, – ограниченность модели, соответствующей первичному запуску и разбору приложения. В качестве дополнительных недостатков можно выделить следующие:

- в процессе составления модели графического интерфейса программы для каждого активного элемента интерфейса и действия над данным элементом создается информация только о том, на какие окна приложения влияет данный элемент. В то же время действия над элементами могут не только закрывать и открывать новые окна, но и создавать, удалять, активировать и деактивировать отдельные графические элементы;
- при работе компонентов Ripper и Replayer используются простые эвристики для сопоставления окон приложения по их заголовкам. При использовании значительного количества диалоговых окон в программе механизм идентификации осуществляет сопоставление окон и модели некорректно.

Для исправления указанных ограничений были разработаны следующие модификации инструмента GUITAR, нацеленные на повышение полноты и точности анализа:

- итеративное построение и обновление модели графического интерфейса и графа потока событий;
- улучшение точности алгоритмов сопоставления элементов графического интерфейса с моделью при запусках тестовых наборов;
- расширение набора параметров отдельных элементов графического пользовательского интерфейса, описывающих эффект от взаимодействия с данными элементами на модель интерфейса.

4. Итеративный анализ программ с графическим пользовательским интерфейсом

Для реализации первого из указанных выше модификаций инструмента GUITAR была предложена альтернативная схема работы компонентов данного инструмента (рис. 2).

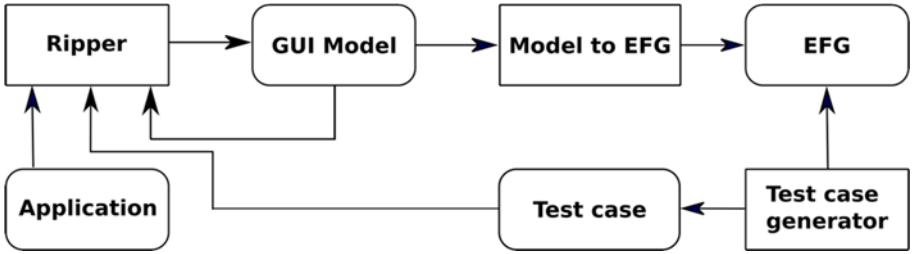

Рис. 2. *Изменение схемы работы GUITAR*

Fig. 2. *GUITAR scheme modification*

Компонент Replayer был исключен из модифицированной схемы взаимодействия инструмента GUITAR. Функциональность данного компонента, включающая возможности воспроизведения последовательности событий, была перенесена в компонент Ripper. Во время работы новой версии компонента Ripper при воспроизведении некоторой последовательности регистрируются все ранее не обнаруженные эффекты выполнения действий над элементами графического пользовательского интерфейса, и информация о них заносится в текущую модель структуры графического интерфейса. В том случае, когда при воспроизведении последовательности открывается ранее не проанализированное окно, компонент Ripper переключается в обычный режим и производит полный анализ данного окна.

Использование модифицированной схемы работы инструмента позволило также повысить точность определения параметров графических элементов и побочных эффектов, соответствующих действиям с данными элементами. Так, например, характеристики элемента, соответствующие возможным типам действий над этим элементом, определяются с большей точностью, чем ранее. В частности, элементы, которые позволяют осуществлять закрытие текущего

окна, могут быть однозначно определены. Ранее в инструменте GUITAR для определения подобных элементов использовалось следующее правило — если тип элемента и текст, соответствующий элементу (например, кнопка и заголовок кнопки) присутствуют в списке терминальных элементов, то действие, соответствующее данному элементу, вызывает закрытие текущего окна. Список терминальных элементов было необходимо задавать вручную перед началом анализа.

4.1 Изменение параметров модели графического пользовательского интерфейса.

Для эффективного применения итеративного анализа в модель графического пользовательского интерфейса программы были внесены следующие изменения.

- Во-первых, был изменен принцип идентификации различных окон. Ранее в инструменте GUITAR идентификация происходила исключительно по заголовку окна. Подобный способ идентификации не позволял эффективно различать окна в некоторых ситуациях (в частности, для стандартных диалоговых окон определённого типа, таких как сообщения об ошибке и информационные сообщения, заголовки которых совпадают). Для разрешения конфликтов в данной ситуации было принято решение проводить сравнение двух окон по заголовку и информации об элементах в составе данного окна. Эти параметры определяют идентификатор окна (к строке, составленной из заголовка и идентификаторов вложенных элементов, применяется хеш-функция, определяющая целочисленное значение идентификатора). Это изменение позволило добиться повышения полноты информации о графической структуре исследуемого приложения.
- Во-вторых, для графических элементов модели был расширен набор характеристик, описывающих эффекты взаимодействия с данными элементами. В частности, были добавлены следующие параметры.
    - Параметр DISABLE_LIST – список элементов, которые становятся неактивными после выполнения действия над ними. Неактивное состояние соответствует невидимым элементам (скрытым встроенными средствами взаимодействия с графическими элементами) или так называемым «выключенным» элементам, которые видимы, но недоступны пользователю. Неактивное состояние означает, что совершение действий над элементом невозможно.

    - Параметр ENABLE_LIST – список элементов, которые становятся активными после выполнения действия над ними.
    - Параметр DESTROY_LIST – список элементов, которые удаляются из содержащего их контейнера после выполнения действия над ними.
    - Параметр CREATE_LIST – список элементов, которые создаются и добавляются в один или различные контейнеры после выполнения действия над ними.
- Для существующих характеристик элементов модели графического интерфейса были произведены следующие модификации:
    - параметр EFFECT_TYPE, определяющий тип воздействия события, связанного с элементом, на общее состояние программы, был дополнен значением SYSTEM_EXIT, соответствующим попытке завершить работу приложения с помощью вызова функции `System.exit()`;
    - возможные значения параметра INVOKE, определяющие список окон, которые открываются при совершении действия над элементом, были изменены со списка заголовков окон на список идентификаторов окон.

Дополнительно была изменена концепция работы с элементами, поддерживающими несколько возможных типов действий. В исходной версии инструмента GUITAR каждому элементу в модели графического интерфейса соответствовало единственное возможное действие. Для повышения полноты анализа был реализован механизм определения всех возможных действий, выполнение которых допустимо для элемента (например, длительное нажатие на кнопку для приложений на платформе Android, которое заменяет двойной щелчок). Во время анализа приложения компонентом Ripper происходит выполнение всех возможных действий для каждого элемента и в модели сохраняется информация о побочных эффектах для пар ⟨ элемент, тип действия⟩.

4.2 Направленная генерация тестовых сценариев

Для осуществления генерации тестовых сценариев в рамках схемы итеративного анализа с помощью инструмента GUITAR разработан модуль UndefinedComponentPath. Этот модуль предназначен для построения последовательностей событий на основе поиска «нерассмотренных элементов», т. е. тех элементов, для которых компонент Ripper не осуществил все возможные действия на предыдущих итерациях анализа. UndefinedComponentPath рассматривает в качестве входных данных граф потока событий. Соответственно, построение последовательностей для достижения «нерассмотренных элементов» сводится к задаче поиска путей в

графе до непосещенных вершин. В графе имеется набор начальных вершин, соответствующий действиям над элементами, которые можно произвести непосредственно после запуска приложения. Искомый путь для достижения «нерассмотренных элементов» должен исходить из одной из начальных вершин. Для облегчения процедуры поиска в граф вводится дополнительная вершина, которая объявляется единственной начальной и из которой имеются переходы в начальные вершины исходного графа.

Однако, граф потока событий не отражает полную информацию, полученную на этапе построения модели. В частности, в переходах никак не учитываются параметры ENABLE_LIST, CREATE_LIST, DISABLE_LIST и DESTROY_LIST, которые могут непосредственно влиять на достижимость некоторого элемента. Рассмотрим простой пример: в приложении имеется три элемента **A**, **B** и **C**, содержащиеся в одном контейнере (например, три кнопки), которым соответствуют действия **x**, **y** и **z**. Так как элементы содержатся в одном контейнере, то в графе потока событий имеются все возможные переходы между вершинами, соответствующими **x**, **y** и **z**. При этом, однако, при выполнении действия **x** над элементом **A**, элемент **C** переводится в неактивное состояние и действие **z** не может быть выполнено. Выполнение действия **y** над элементом **B**, напротив, переводит элемент **C** в активное состояние. Это означает, что последовательности событий, в которых действие **z** находится непосредственно после действия **x**, не являются допустимыми относительно действия **z**, так как это действие не может быть выполнено в связи с тем, что элемент **C** находится в неактивном состоянии. Напротив, в том случае, если элемент **C** изначально неактивен (то есть неактивен при запуске приложения), допустимыми относительно действия **z** являются только пути, содержащие действие **y,** переводящее элемент **C** в активное состояние.

Для непосредственной реализации модуля был выбран алгоритм Флойда-Уоршелла [6], который позволяет рассчитать кратчайшие пути между всеми вершинами рассматриваемого графа. Далее, на основе параметров ENABLE_LIST, CREATE_LIST, DISABLE_LIST и DESTROY_LIST вершин графа производится уточнение найденных путей.

Рассмотрим следующие модификации путей. Пусть путь содержит вершины **X** и **Y**, которые соответствуют элементам **A** и **B**, и выполняются следующие свойства:

- элемент **B** содержится в DISABLE_LIST(**X**) или DESTROY_LIST(**X**);
- событие **Y** находится в последовательности после события **X;**
- для всех событий **Z**, которые находятся между **X** и **Y**, элемент **B** не содержится ни в ENABLE_LIST(**Z**), ни в CREATE_LIST(**Z**).

Данный путь является недопустимым для элемента **B** и необходимо произвести разбиение пути на составляющие, не содержащие событие **X**.

Пусть путь содержит вершину **Y**, которая соответствуют элементу **B**, причем справедливо:

- элемент **B** содержится в ENABLE_LIST(**X**) или в CREATE_LIST(**X**);
- элемент **B** либо ещё не создан при запуске приложения, либо находится в неактивном состоянии**.**

Данный путь является недопустимым для элемента **B** и необходимо произвести разбиение пути на три составляющие:

- от начальной вершины до вершины **X**;
- от вершины **X** до вершины **Y**;
- от вершины **Y** до целевой вершины.

Использование компонента UndefinedComponentPath, использующего описанный выше метод поиска путей в графе потока событий и уточнение найденных путей, позволяет последовательно исследовать необработанные элементы графического пользовательского интерфейса. Каждый запуск программы на выполнение, нацеленный на обработку одного из элементов, позволяет обновить модель интерфейса, граф потока событий и увеличить полноту описания графического интерфейса.

5. Результаты применения инструмента

Исследование эффективности предложенного подхода проводилось на наборе тестовых приложений и наборе реальных проектов. Рассматриваемые проекты выбирались из списка приложений, которые ранее анализировались инструментом GUITAR. Приведём краткое описание данных проектов:

- JabRef – менеджер научных публикаций;
- Rachota – планировщик рабочего времени;
- TippyTipper – средство на платформе Android для расчёта чаевых и распределения платы на несколько человек;
- Browser – приложение операционной системы Android для просмотра веб-страниц.

Использование итеративного анализа с обновлением модели структуры графического интерфейса приложения позволило увеличить полноту данной модели.

Результаты применения итеративного анализа приведены в следующей таблице (в левой части представлены параметры модели, создаваемой при помощи оригинальной версии инструмента, в правой части — параметры модели, создаваемой с помощью модифицированной версии GUITAR после обхода всех необработанных элементов):

Табл. 1. Результаты применения итеративного анализа
Table 1. Results of applying iterative analysis

Программы	GUITAR			Iterative Ripper		
	число окон	число элементов	число событий	число окон	число элементов	число событий
Rachota	11	455	166	13	512	201
JabRef	42	1301	706	48	2005	1201
TippyTipper	3	45	24	5	58	34
Browser	3	43	27	5	51	31

Применение модифицированной схемы инструмента позволило построить более точную и полную модель графического интерфейса, которая может быть использована для создания тестовых сценариев с целью проверки корректности работы программ.

6. Заключение

В данной статье рассмотрена задача автоматизации динамического анализа приложений, предоставляющих графический пользовательский интерфейс. Среди существующих инструментов, применяемых в промышленной разработке программного обеспечения и в академических исследованиях, было выбрано средство GUITAR, обеспечивающее максимальную степень автоматизации процесса построения тестовых сценариев для проверки работы программы. На основе понятий модели представления структуры графического интерфейса и графа потока событий, введенных авторами инструмента GUITAR, была предложена схема итеративного анализа программы, позволяющая последовательно расширять и дополнять модель пользовательского интерфейса программы и строить новые тестовые наборы для обеспечения наиболее полного покрытия кода программы.

Предложенная схема показала лучшие результаты, чем оригинальная схема инструмента GUITAR, на всех исследованных приложениях с открытым исходным кодом. Выигрыш по эффективности анализа был достигнут за счёт создания модели, более точно описывающей реальную структуру графического пользовательского интерфейса. Теоретически показано, что большая точность модели позволит уменьшить количество тестовых наборов, за счет исключения тестовых сценариев, невозможных при реальной работе с приложением.

Стоит отметить, что в подходе к проведению автоматического динамического анализа программ, предоставляющих графический пользовательский интерфейс, предложенном авторами инструмента GUITAR и расширенном в

рамках исследования, существует ряд практических и теоретических ограничений. В частности, даже для приложений с малым количеством элементов графического интерфейса возникает проблема большого количества возможных тестовых наборов и выбора наиболее приоритетных тестовых наборов, позволяющих минимизировать время анализа, сохраняя полноту покрытия кода программы. В настоящее время активно развиваются подходы, связанные с интеллектуальным обходом путей выполнения в программе, использующие концепции символьного исполнения, статического анализа исходного и исполняемого кода. Исследования, связанные с возможностями совмещения методов, базирующихся на модели графического интерфейса, и указанных выше методов, представляются крайне перспективными в свете высокой востребованности автоматических инструментов оценки качества программного обеспечения.

Список литературы

[1]. Страница проекта Ranorex. http://www.ranorex.com/ [HTML]. Обращение от 10.10.2016
[2]. Abbot framework for automated testing of Java GUI components and programs. https://abbot.sourceforge.net. Обращение от 10.10.2016
[3]. Страница проекта Maveryx. https://sourceforge.net/projects/maveryx/ [HTML]. Обращение от 10.10.2016
[4]. Страница проекта Squish. https://www.froglogic.com/squish/ [HTML]. Обращение от 10.10.2016
[5]. Bao N. Nguyen, Bryan Robbins, Ishan Banerjee, Atif Memon. GUITAR: an innovative tool for automated testing of GUI-driven software. Automated software engineering, 2013, vol. 21(1), pp. 65-105.
[6]. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ, 2-е изд. М.:«Вильямс», 2006. 1296 с.

Applying iterative dynamic analysis to programs with graphical user interface

M.K. Ermakov <mermakov@ispras.ru>
A.Y. Gerasimov <agerasimov@ispras.ru>
D.O. Kutz <kutz@ispras.ru>
A.A. Novikov <a.novikov@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. This paper is dedicated to practical research in the field of automated testing and analysis of software that features a graphical user interface. Current tendencies in user interface development favour semi-automatic approaches that employ human experts to create and prepare test suites. The ever-increasing complexity of software leads to decreased effectiveness of these approaches, especially when one considers large amounts of computational resources

available during development. We present a fully automatic approach to dynamic analysis of program graphical interfaces. Our approach is based on the open-source GUITAR tool, which we have identified among other industrial and academic tools as the one closest to full automation. While highly efficient, GUITAR nevertheless has certain drawbacks and limitations which might cause insufficient accuracy in modelling the graphical interface structure and its individual elements. In turn, these limitations lead to fragments of graphical interface not getting processed during analysis or cause incorrect (i.e. not reproducible in practice) test cases to be generated. Our contributions include a set of modifications: incremental graphical interface model generation, improved identification of graphical interface element attributes and side effects, and finally a test case generation algorithm that focuses on reaching unprocessed graphical interface elements to check their functionality and improve the model. We have tested our modifications on a set of open source projects originally checked by GUITAR developers and achieved positive results: increased precision of GUI structure model and theoretically can decrease number of inapplicable test cases. Finally, we discuss several potential improvements for future work, including, in particular, the use of dynamic symbolic execution methods.

Keywords: dynamic analysis; program analysis; GUI testing; test coverage.

DOI: 10.15514/ISPRAS-2017-29(1)-8

For citation: Ermakov M.K., Gerasimov A.Y., Kutz D. O., Novikov A.A. Applying iterative dynamic analysis of programs with graphical user interface. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 1, 2017, pp. 119-134 (in Russian). DOI: 10.15514/ISPRAS-2017-29(1)-8

References

[1.] Ranorex. http://www.ranorex.com/ [HTML]. Accessed at 10.10.2016

[2.] Abbot framework for automated testing of Java GUI components and programs. https://abbot.sourceforge.net. Accessed at 10.10.2016

[3.] Maveryx. https://sourceforge.net/projects/maveryx/ [HTML]. Accessed at 10.10.2016

[4.] Squish. https://www.froglogic.com/squish/ [HTML]. Accessed at 10.10.2016

[5.] Bao N. Nguyen, Bryan Robbins, Ishan Banerjee, Atif Memon. GUITAR: an innovative tool for automated testing of GUI-driven software. Automated software engineering, 2013, vol. 21(1), pp. 65-105.

[6.] Tomas H. Kormen, Charl'z I. Lejzerson, Ronal'd L. Rivest, Klifford Shtajn. Introduction to Algorithms, 2-e izd. M.:«Vil'jams», 2006. 1296 p. (in Russian)