Автоматизация регрессионного тестирования при помощи анализа трасс событий[1]

Владимир Федотов
Институт системного программирования РАН
e-mail: vfl@ispras.ru

Аннотация. В настоящей работе представлен process mining алгоритм, предназначеный для применения в инструментированной распределенной системе. Подразумевается, что система инструментирована таким образом, что взаимодействия между модулями системы проходят через единую шину, связывающую все компоненты системы. Записанные при обработке на шине события анализируются представленным в работе алгоритмом, результатом работы которого является модель взаимодействий между модулями системы. Полученная модель используется для разработки покрывающего регрессионного тестового набора.

Ключевые слова: регрессионное тестирование; распределенные системы; process mining

1 Введение

Одним из трендов развития современных информационных систем является переход к большей распределенности компонентов этих систем. Код этих компонентов выполняется не просто на разных процессорах, но и на разных машинах. Эти компоненты не имеют общей памяти и, все чаще, не имеют общего хранилища данных. Взаимодействия между ними построены на обмене сообщениями поверх различных протоколов, среди которых чаще других используется HTTP.

Причины, по которым архитектура систем стремится к распределенности, разнообразны. Это и популярность Agile разработки с ее короткими итерациями, которые легче подстраивать под небольшие модули системы. Это и дешевизна современных средств виртуализации. Это и существенно возросшая скорость обмена данными через

Работа поддержана ФЦП "Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2013 годы" (контракт N 11.519.11.4024).

локальные сети и интернет. Это и современное стремление бизнеса к глобализации, которое ведет к тому, что в один бизнес-процесс могут быть вовлечены IT-системы нескольких компаний.

Другой тенденцией, без сомнения связанной с первой, является уменьшение функциональной значимости каждого компонента в отдельности. Зачастую один компонент выполняет единственную, достаточно тривиальную, функцию – например, маршрутизацию входящих запросов на другие компоненты системы по определенным правилам.

С точки зрения разработки, такие компоненты легче поддерживать, так как их код относительно тривиален и не превышает объемом 1 kloc. Главное же преимущество такого подхода состоит в масштабируемости, которая имеет первостепенную важность для современной информационной системы. Так, система, состоящая из набора модулей, не связанных общим хранилищем данных, масштабируется линейно.

Внедрение подобной архитектуры приводит к смещению акцентов при тестировании как отдельных модулей, так и всей системы вцелом. С одной стороны юнит-тестирование теряет значимость, так как каждый компонент в отдельность тривиален. С другой стороны, интеграционное и системное тестирование приобретают особую важность, так как именно проблемы взаимодействия компонентов являются наиболее критическими для работы распределенной системы.

Большая часть взаимодействий происходит поверх стандартных интернет-протоколов, таких как HTTP, которые не предполагают никакой типизации передаваемых объектов. Таким образом, все ошибки типов и форматов данных обнаруживаются лишь в run-time и лишь на этапе интеграционного тестирования.

Другим следствием распределенности системы является то, что документация также становится все более «распределенной». Описание фактически единого бизнес-процесса либо разбивается на спецификации множества независимых модулей, либо отсутствует вовсе. Первой жертвой этой проблемы становится регрессионное тестирование, для обеспечения которого необходима полная и актуальная документация.

Негативно влияет на качество регрессионного тестирования и сама природа распределенной системы. Обмен сообщениями между модулями подразумевает, что многие логические зависимости в системе являются семантическими зависимостями между передаваемыми данными. Так, например, строки 'aaaa' и 'bbbb' могут представлять различные классы экививалентности для вызываающей системы в зависимости от того, какие атрибуты связаны с ними в вызываемой системе.

Определить эти классы эквивалентности по спецификации вызывающей системы невозможно. Более того, функциональные спецификации зачастую не содержат описания семантических связей данных или

описывают их как «корректные» и «некорректные», не раскрывая смысл этих понятий.

2 Data-mining в целях тестирования

Прямым следствием отсутствия полной документации и сложности поиска семантических зависимостей является то, что для создания полноценных регрессионных тестовых наборов тестировщикам все чаще приходится прибегать к реверс-инжинирингу требований, выполняемому на предыдущей версии системы.

Впрочем, подобный ad-hoc метод также несет в себе значительные риски, так как не предоставляет никаких гарантий полноты полученного набора требований.

Более перспективным методом является run-time анализ поведения существующей системы в продуктивном окружении. Такой анализ возможен благодаря тому, что большинство коммерческих систем снабжены средствами мониторинга и отчетности (так называемый business intelligence), записывающими обрабатываемые системой события.

Хорошие результаты также приносит анализ системных журналов. Однако в связи с их объемом и неупорядоченностью данных такой анализ требует применения специальных средств и методов.

В действительности, анализ данных, или data mining является черезвычайно популярной темой исследований. Методы data mining работают как в уже упомянутых средствах business intelligence, так и применяются ad hoc для анализа неподготовленных данных.

Цели анализа могут быть различными: поиск статистических зависимостей, маркетинговые исследования, анализ производительности системы [2], [5]. В некоторых случаях методы data mining применимы даже для анализа эффективности дизайна нового сайта (сколько кликов сделал пользователь, прежде чем нашел желаемое) и поиска угроз безопасности системы [7].

Методы data minig также применимы в тестировании, прежде всего именно для run-time анализа системы. Так, в [6] описан перспективный подход к построению регрессионных тестовых наборов и поиску классов экивалентности данных при помощи анализа трасс событий в системе.

В целом, такие подходы, позволяющие восстановить модель принятия решений системой, выделились в отдельный класс, получивший название «process mining» [8].

Для нас в первую очередь интересно примение методов process mining для восстановления модели системы, применимой для регрессионного тестирования. Перспективный подход описан в [1] как дельта-анализ двух моделей системы (например, действующей и доработанной версий) и тестирование соответствия модели фактической реализации системы и эталонной модели системы.

Впрочем, подход описывает лишь общие принципы такого тестирования и не подходит для внедрения в виде законченной методологии тестирования.

Отдельно стоит рассмотреть алгоритмы, используемые в подходах process mining. На настоящий момент они разбиваются на общие и специфические. Специфические алгоритмы эффективны на узком классе данных. Общие алгоритмы эффективны на всех классах данных, однако гораздно менее эффективны, чем специфические алогритмы, на конкретном классе данных.

Таким образом, применение process mining для строго определенной задачи подразумевает либо разработку нового спецического алгоритма, либо поиск наиболее подходящего из существующих.

Критерием эффективности такого алгоритма является сравнение с одним из общих алгоритмов. В качестве эталонного алгоритма принято выбирать α -алгоритм, описанный в [8].

В настоящей работе мы описываем алгоритм process mining, применимый для получения регрессионных тестовых наборов, и основные методы его применения.

3 Process-mining событий в ESB

Для применения mining-подходов первоочередное значение имеет способ, которым собираются события для анализа. Мы применяем подход к инструментации тестового окружения при помощи открытых ESB-платформ, описанный в [3] и [4]. Инструментированное подобным образом окружение позволяет собирать трассу сообщений, представленных HTTP-запросами и ответами компонентов системы.

3.1 Свойства сообщений

В нашем подходе мы абстрагируемся от синхронности или асинхронности анализируемых взаимодействий, поэтому для выделения зависимостей (корелляции) между сообщениями все проходящие через шину события дискретизируются.

На практике это означает, что все попавшие на шину события собираются в единую очередь. Обработчик очереди запускается через равные промежутки времени, представляющие собой период дискретизации. При обработке очередного сообщения к нему добавляется текущее значение счетчика периодов или, проще говоря, шаг, на котором оно было обработано.

Таким образом, обработанное сообщение записывается в трассу с указанием 1) шага обработки; 2) компонента-отправителя; 3) компонента-получателя. Также записываются поля сообщения и их значения. Подразумевается, что сообщение представимо в виде XML-

документа (1). Таким образом, поля сообщения являются листьями документа, значения полей – текстовыми значениями листьев.

$$m = \{t, src, dst, F, V\} \quad (1)$$

Дискретизация времени обработки сообщений влечет за собой интересные следствия. В частности, существенно упрощается анализ зависимостей между сообщениями. Так, сообщения, где $m_1.t = m_2.t$, независимы, если $m_1.src \neq m_2.src$. В противном случае, это означает, что отправитель выполняет два параллельных запроса.

3.2 Корелляция сообщений

Применяемый нами подход основан на поиске зависимых сообщений в трассе событий. Так, например, зависимыми могут быть пара запрос-ответ или запрос, порождающий последующий запрос. В различных BPEL-инструментах принято использовать так называемый идентификатор корелляции, который передается по цепочке от предыдущего сообщения следующему, однозначно идентифицируя, таким образом, зависимость. Разумеется, в общем случае рассчитывать на наличие такого идентификатора не приходится.

Для определения отношения корелляции сообщений мы используем сравнение основных параметров сообщения, как показано в (2). Таким образом, сообщения зависимы, только если отправитель последующего сообщения является получателем предыдущего, и шаг обработки различается строго на 1.

$$m_1 \rightarrow m_2 \Leftarrow m_2.src = m_1.dst \wedge m_2.t - m_1.t = 1 \quad (2)$$

Отношение (2) является верным (свидетельствует о корелляции) во всех случаях, где дискретизация сообщений выполнена корректно, то есть зависимые сообщения обработаны на 1) разных шагах обработки; 2) последовательных шагах обработки.

3.3 Интеракции

Говоря о корелляции сообщений, логично рассматривать цепочку кореллирующих сообщений, как единую сущность. В BPEL каждая такая цепочка является результатом выполения процесса для каждого конкретного входящего запроса или тестового воздействия. Само существование и длина цепочек сообщений обусловлены распределенностью исследуемых систем. Структура цепочек может различаться в зависимости от используемых в системе транспортных

протоколов. Так, например, при использовании синхронного HTTP взаимодействие в системе из трех модулей будет представлено цепочкой «вложенных» взаимодействий:

$$Client \rightarrow A \rightarrow B \rightarrow C \rightarrow B \rightarrow A \rightarrow Client.$$

В данном случае синхронность подразумевает, что модуль A удерживает HTTP-сессию с клиентом открытой до тех пор, пока остальные модули системы не завершат работу. Цепочку, подобную этой, мы называем интеракцией. Каждая интеракция состоит из попарно кореллирующих сообщений: $i = m_1 \rightarrow m_2 \rightarrow \ldots \rightarrow m_n$

Таким образом, трасса, собраная в ходе работы системы, представляет собой набор интеракций. Каждая интеракция состоит из сообщения-стимула и последующих, кореллирующих друг с другом сообщений (3). Вырожденная интеракция состоит только из сообщения-стимула, представляя собой случай, когда никакой реакции не последовало.

$$i = \{m_0, M\} \quad (3)$$

Конечным сообщением в интеракции является сообщение, на которое не последовала реакция на следующем шаге обработки. Конечное сообщение может совпадать с начальным или не быть единственным в случаях, когда один из модулей системы генерирует несколько параллельных запросов.

В общем виде, интеракция является деревом, корень которого обозначает сообщение-стимул, а листья являются конечными сообщениями. Интеракция, в которой все листовые сообщения являются конечными, помечается как закрытая.

3.4 Свойства интеракций

Мы рассматриваем интеракции как прообраз тестов. Так, m_0 является тестовым воздействием, а M – ожидаемой реакцией. Можно сказать, что отношение $M_x ? My, m_0^x = m_0^y$ является автоматическим тестовым оракулом.

Разумеется, каждая трасса может содержать произвольное количество идентичных интеракций. Поэтому использование для тестов всей трассы нецелесообразно. Мы анализируем идентичность интеракций и составляющих их сообщений на нескольких уровнях для того, чтобы отсечь дублирующие друг друга интеракции.

Для определения степени равенства двух сообщений мы используем три различных по строгости отношения. Так, отношение эквивалентности подразумевает равенство всех атрибутов сообщений (4). Отношение равенства подразумевает равенство всех адресных атрибутов

сообщений, а также их структуры (5), данные в сообщениях могут различаться. Наконец, отношение схожести подразумевает только равенство всех адресных атрибутов сообщений (6). Отношение экивалентности является наиболее сильным, отношение схожести наиболее слабым: $m_x = m_y \Rightarrow m_x;m_y \Rightarrow m_x \approx m_y$

$$m_x = m_y \Leftarrow m_x.t = m_y.t \wedge m_x.src = m_y.src \wedge m_x.dst = m_y.dst \wedge (\forall f_x : f_x = f_y \wedge v(f_x) = v(f_y)) \quad (4)$$

$$m_x;m_y \Leftarrow m_x.t = m_y.t \wedge m_x.src = m_y.src \wedge m_x.dst = m_y.dst \wedge (\forall f_x : f_x = f_y) \quad (5)$$

$$m_x \approx m_y \Leftarrow m_x.t = m_y.t \wedge m_x.src = m_y.src \wedge m_x.dst = m_y.dst \quad (6)$$

Из отношения равенства вложенных сообщений следует отношение равенства интеракций. Так как интеракция может содержать сообщения, состоящие в разных степенях равенства, отношение равенства интеракций соответствует наиболее слабому из отношений равенства вложенных сообщений. Правила (7, 8, 9) определения равенства двух интеракций представлены для интеракций из двух сообщений и аналогичны для интеракций из большего числа сообщений.

$$i_x = i_y \Leftarrow \forall m_x : m_x = m_y \quad (7)$$

$$i_x;i_y \Leftarrow \forall m_x : m_x;m_y \vee i_x = i_y \quad (8)$$

$$i_x \approx i_y \Leftarrow \forall m_x : m_x \approx m_y \vee i_x = i_y \vee i_x;i_y \quad (9)$$

3.5 Алгоритм формирования модели интеракций

Нулевым шагом алгоритма является формирование интеракций. Формирование интеракций происходит непосредственно при обработке очередного сообщения. Если сообщение кореллирует с одной из существующих интеракций, эта интеракция дополняется новым сообщением. Если же нет, сообщение инициирует создание новой интеракции. Ситуация, при которой сообщение кореллирует более чем с одной незакрытой интеракцией, невозможна. Завершение сбора трассы

означает, что новые интеракции перестают добавляться в трассу, все незакрытые интеракции помечаются как закрытые.

Далее из трассы удаляются все дублирующиеся интеракции, такие что $i_1 = i_2 \vee i_1;i_2$

Результирующий набор интеракций трансформируется в направленный ациклический граф следующим образом: для каждой последующей интеракции каждое последующее сообщение является новым узлом графа, если граф не содержит другого узла, для которого выполнялось бы условие $m_x;m_y$. Для каждого узла достижимыми являются узлы, представленные коррелирующими сообщениями.

Трасса также может содержать интеракции, все сообщения в которых не кореллируют друг с другом. Такие интеракции являются независимыми и относятся к разным моделям. Количество сформированных моделей по итогам обработки трассы равно количеству попарно независимых интеракций в ней.

3.6 Расширяемость моделей

Модели, полученные в результате обработки трассы, расширяемы. Это означает, что модель, полученная при обработке одной трассы, может быть дополнена моделью, полученной при обработке другой трассы. Модель M_1 расширяема моделью M_2, если в модели M_1 существует хотя бы одно сообщение, для которого выполняется отношение равенства для любого другого сообщения из модели M_2 (10).

$$\exists m_x(M_1) : m_x;m_y(M_2) \quad (10)$$

3.7 Генерация тестов

Конечной целью майнинга событий в системе является генерация тестового набора, покрывающего модель взаимодействий в системе. Такой тестовый набор может использоваться в дальнейшем, как регрессионный или как промежуточный, для взаимосвязи различных этапов тестирования за счет расширяемости получаемых моделей.

Тестовый набор, сгенерированный на основе модели, состоит из набор тестовых воздействий, представляющих собой все начальные узлы модели, и оракула, представленного остальными узлами модели. В данном случае мы исходим из того, что в ходе выполнения полученных тестов будет сгенерирована новая модель взаимодействий, сравнение которой с эталонной (предыдущей) моделью дает представление о различиях в поведении системы.

В целом, критерием корректности считается следующее: тест T является пройденным, если в результате тестового воздействия m_0 порождается интеракция I, такая что: $\forall m_x(T)\exists m_y(I) : m_x; m_y$

3.8 Предположения и ограничения

Основные ограничения подхода следуют из используемого отношения зависимости (2).

Подразумевается, что анализируемая система реализует event-driven архитектуру. То есть сообщения не могут возникать в ней спонтанно, например, вследствие срабатывания таймера. Каждое сообщение в системе появляется лишь после 1) обработки предыдущего сообщения; 2) пользовательского воздействия.

Описанное отношение зависимости (2) имеет смысл только для систем взаимодействия, которые построены на обмене сообщениями. Оно не имеет смысла для систем, обменивающихся файлами или транзакциями. Так, например, подход неприменим для ETL-систем.

Подразумевается, что все зависимые сообщения обрабатываются в рамках периода дискретизации системы. Таким образом, если время обработки запроса превышает период дискретизации, то ответное сообщение не будет считаться результатом обработки запроса. Впрочем, в реальных системах, при периоде дискретизации равному половине HTTP-таймаута, влияние этого ограничения на наш взгляд минимально.

4 Заключение

В настоящей работе был представлен специфический алгоритм process mining, который предназначен для применения в инструментированной распределенной системе. Подразумевается, что система инструментирована таким образом, что взаимодействия между модулями системы проходят через единую шину, где дискретизируются и записываются. Записанные таким образом события анализируются представленным в работе алгоритмом, результатом работы которого является модель взаимодействий между модулями системы. Полученная модель используется для разработки покрывающего регрессионного тестового набора.

В последующих работах будет представлено сравнение результатов разработанного алгоритма и generic алгоритма, в качестве которого используется α-алгоритм. Также мы планируем подготовить описание метода применения разработанного алгоритма для внедрения 1) в процесс регрессионного тестирования распределенной системы; 2) в стандартный, построенный в соответствии с V-моделью процесс тестирования. Там же будут представлены примеры применения и результаты апробации подхода на реальных проектах.

В долгосрочной перспективе мы планируем разработать полноценный набор средств для инструментации распределенной системы. Предложенный в этой работе подход будет дополнен методом выделения классов эквивалентности данных и их внедрения в тестовую модель. Также мы планируем разработать набор эвристик для поиска скрытых состояний в распределенной системе. Такие эвристики позволят выявлять и внедрять в модель события, изменяющие состояние того или иного объекта в системе.

Список литературы

[1] W. M. P. Van Der Aalst. Business alignment: using process mining as a tool for Delta analysis and conformance testing. *Requirements Engineering*, 10(3):198–211, August 2005.

[2] A Adriansyah and JCAM Buijs. Mining Process Performance from Event Logs. *Business Process Management Workshops*, 2013.

[3] Vladimir Fedotov. Run-time monitoring for model-based testing of distributed systems. In *SYRCoSE 2012*, 2012.

[4] VN Fedotov. Service-oriented approach to integration testing in distributed systems. *SYRCoSE 2010*, page 58, 2010.

[5] Rattikorn Hewett. Mining software defect data to support software testing management. *Applied Intelligence*, 34(2):245–257, September 2009.

[6] Mark Last, Menahem Friedman, and Abraham Kandel. The data mining approach to automated software testing. *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '03*, page 388, 2003.

[7] Jianxiong Luo and SM Bridges. Mining fuzzy association rules and fuzzy frequency episodes for intrusion detection. *International Journal of Intelligent Systems*, 15(I):1–36, 2000.

[8] W. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, September 2004.

Automated event trace analysis for regression testing

Vladimir Fedotov

Abstract. Modern IT systems become more and more distributed, both in literal sense - as business logic spreads across applications running in several different network domains, and in a metaphorical sense - as expert knowledge about system's inner working spreads across different outsource companies and hundreds of documents. In this paper we are discussing regression testing as one of the issues related to that trend.

When it comes to testing, quality of specifications is a major issue. They are either unmanageably big and outdated, or very much fragmented - each specifying its own module, but lacking perspective on how system works as a whole. In our practice, it often comes down to reverse engineering to discover what exactly different parts of the system are expecting from each other, but such ad hoc techniques are not welcome in the domain of testing.

Different approach relies on analyzing system's events trace that is provided either by business activity monitoring tools, transaction logs or, simply, log files. There is a lot of different data mining techniques already developed; we are mostly interested in process mining as a way of discovering system's decision model.

A key factor of any process mining application is the algorithm of discovering events relation. It can be either generic algorithm like Alpha-algorithm or a specific algorithm tailored for a particular set of possible events. In this paper we provide an outline for a specific algorithm that we use to discover events relation in a special environment.

The environment we are using is specifically designed for the goal and has central part in the process. It is used for mediating interactions happening between system's modules in run-time as a discrete process thus providing us with a very straightforward way of determining events causality.

Keywords: regression testing; distributed systems; process mining

References

[1]. W. van der Aalst. Business alignment: using process mining as a tool for Delta analysis and conformance testing. Requirements Engineering, 10(3):198–211, August 2005.

[2]. A. Adriansyah and J. Buijs. Mining Process Performance from Event Logs. Business Process Management Workshops, 2013.

[3]. V.N. Fedotov. Run-time monitoring for model-based testing of distributed systems. SYRCoSE 2012.

[4]. V.N. Fedotov. Service-oriented approach to integration testing in distributed systems. SYRCoSE 2010.

[5]. R. Hewett. Mining software defect data to support software testing management. Applied Intelligence, 34(2):245–257, September 2009.

[6]. M. Last, M. Friedman, A. Kandel. The data mining approach to automated software testing. Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '03, page 388, 2003.

[7]. J. Luo, S.M. Bridges. Mining fuzzy association rules and fuzzy frequency episodes for intrusion detection. International Journal of Intelligent Systems, 15(I):1–36, 2000.

[8]. W. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering, 16(9):1128–1142, September 2004.