Технология создания гетерогенных трасс, их анализа и генерации из них отчётов

С. Г. Грошев
sgroshev@ispras.ru

Аннотация. В статье описывается архитектура модульной системы трассировки (журнализации, логгирования) и анализа трасс, поддерживающей трассировку сложноструктурированной информации, расширение видов трассируемой информации, изоляцию и выборку нужных данных при анализе полученных трасс и облегчающей связывание разнородных данных между собой. Рассматриваются методы связывания данных, в том числе разнородных, по содержанию и по времени, а также дальнейшего преобразования их в разнообразные отчёты. Описывается архитектура генератора отчётов, позволяющая собирать отчёты из независимых модулей и связывать извлечённые ими данные.

1. Введение

Для удовлетворения потребностей пользователей и поддержки стабильного развития современного общества необходимо разрабатывать всё более сложное программное обеспечение (ПО). С ростом сложности ПО увеличивается и сложность его тестирования.

До настоящего времени широко применяются методики тестирования, при которых тесты выдают в конце работы только вердикт вида «PASS» или «FAIL»; при этом предполагается, что в случае неуспешного прохождения какого-либо теста можно легко локализовать и исправить соответствующую ошибку. Однако при исчерпывающем тестировании сложных систем, особенно состоящих из множества взаимодействующих компонентов, этого обычно недостаточно. Поэтому необходимо в ходе тестирования собирать, а после завершения теста анализировать гораздо больше разнообразной информации, например следующей.

- Состояния тестируемой системы в целом или её отдельных компонентов, осуществляемые тестовой системой воздействия на неё и переданные при этом тестовые данные – для анализа ситуаций, в которых возникли ошибки, а также для оценки тестового покрытия;
- Обмен данными между компонентами тестируемой системы;

- Информация о критических секциях и синхронизации параллельно работающих компонентов;
- Сообщения об ошибках (как от самой тестируемой системы, так и от тестов, считающих её поведение некорректным);
- Разнообразные метрики тестового покрытия (по входным данным, по исходному коду, по интерфейсным вызовам, по сценариям использования и так далее) – для оценки полноты тестирования;
- Другие данные, которые могут понадобиться при анализе поведения разрабатываемой системы.

Процесс сбора информации о ходе работы программы и/или тестов для неё называется трассировкой, а собранные данные, упорядоченные по времени возникновения соответствующих событий, – трассой. Отметим сразу, что во многих русскоязычных источниках словом «трассировка» обозначается пошаговая отладка программ (в качестве отражения тенденции можно привести, например, определение слова «Трассировка» в русскоязычной википедии [1]); в настоящей статье под трассировкой понимается то же, что в англоязычных источниках: журнализация (logging) событий, происходящих при работе программы или теста для неё, высокой степени детализации, рассчитанная на то, что какая-то (иногда заранее неизвестно, какая) часть этой информации может понадобиться для анализа поведения этой программы (сравним с определением слова «Tracing» в англоязычной википедии [2]).

Во многих широко используемых сегодня технологиях разработки ПО (например, Rational Unified Process [3], eXtreme Programming [4], а также методиках разработки, известных под общим названием Agile Software Development [5]) разрабатываемая система создаётся итеративно, при этом тесты для неё разрабатываются параллельно с ней, усложняясь по мере её развития. Для оценки правильности поведения тестируемой системы необходимо собирать в ходе тестирования информацию о различных аспектах её поведения, при этом по мере её развития могут добавляться новые виды информации.

Жизненный цикл трассы состоит из двух основных фаз: создание трассы и её анализ; вспомогательные процессы этого жизненного цикла, такие как конфигурирование трассировки, хранение полученных трасс и так далее, в настоящей статье не рассматриваются. На практике фаза анализа может опускаться: в этом случае, например, логи сервера анализируются только в случае обнаружения каких-либо проблем в его работе. Описываемая технология трассировки и анализа трасс разработана для случаев, когда анализ полученных в результате работы целевого приложения (или тестов для него) трасс важен, и основная цель использования системы трассировки – получение неких отчётов в результате анализа трасс. Соответственно, мы рассматриваем нужды двух приложений, использующих описываемую

технологию: генерирующего трассы (также называемого, в зависимости от контекста, целевой системой; в случае тестирования, с точки зрения жизненного цикла трассы целевой системой будет объединение тестовой и тестируемой систем) и анализирующего их (также называемого генератором отчётов).

Как видно из вышесказанного, собираемая в виде трасс информация о поведении целевой системы может быть чрезвычайно разнообразной, причём её состав может часто меняться (в этом её отличие, например, от лога работы веб-сервера, содержащего только сообщения фиксированного формата об обработанных запросах). Такие данные и содержащие их трассы мы называем гетерогенными; они требуют особых способов генерации и анализа.

В настоящей статье рассматриваются:

1. Архитектура системы, предназначенной для создания гетерогенных трасс и облегчающей их анализ.
2. Методы анализа гетерогенных данных.
3. Архитектура генератора отчётов, позволяющего объединять результаты анализа данных, проведённого различными аналитическими модулями, и представлять их в удобном для человека виде.

2. Архитектура системы трассировки и анализа трасс

2.1. Требования

Для решения задач, встающих при работе с гетерогенными трассами, были выделены следующие требования, которым должна удовлетворять система поддержки создания и анализа гетерогенных трасс:

- Универсальность.

Решение не должно быть привязано к какой-либо предметной области, классу приложений, способу тестирования (и вообще к задачам тестирования), языку программирования, платформе или способу хранения данных. Предоставляемая функциональность должна обеспечивать применимость в широком круге задач, в которых имеются аналогичные потребности мониторинга поведения сложных систем и анализа полученных разнородных данных.

- Расширяемость и модульность.

Структура трассы и архитектура инструментов её генерации и обработки должны без существенных накладных расходов допускать добавление новых видов отслеживаемых событий. При этом:

  - Инструменты, анализирующие трассу, должны иметь возможность получать только существенную для них информацию, игнорируя всё остальное эффективно (как по скорости обработки, так и по используемой памяти) и без потери согласованности данных.
  - Необходимы средства, позволяющие легко добавлять функциональность анализа новых видов информации к уже существующим генераторам отчётов. Для решения этой задачи нужна инфраструктура, позволяющая собирать инструменты анализа трасс из слабо связанных анализирующих компонентов и автоматически связывать информацию, собранную этими компонентами, между собой.

- Возможность работы со сложными структурами данных.

Необходима возможность сохранять в трассе информацию о сложных объектах, с которыми работает целевая система, и потом легко восстанавливать их структуру. Эта задача особенно актуальна при тестировании программных интерфейсов, работающих с такими объектами.

- Унификация.

Для снижения трудоемкости разработки инструментов анализа и обеспечения возможности повторного использования их кода, схожие виды информации должны сбрасываться в трассу в одинаковом формате.

В современной программной индустрии уже разработано множество решений для мониторинга работы приложений (особенно хотим отметить SystemTap [6] – проект с открытым кодом, позволяющий динамически инструментировать ядро операционной системы для трассировки выбранных системных событий), однако в большинстве случаев они концентрируются на задачах сбора информации с работающей системы, а не на самой этой информации. При поиске подходящих уже существующих решений мониторинга с возможностью расширения видов собираемых данных были также найдены два проекта, имеющих одинаковое название «Расширяемый формат лога» (XLF, eXtensible Log Format). Однако несмотря на заявленную расширяемость, оба они работали с достаточно примитивными и малорасширяемыми данными: один из них [7] разрабатывался как замена для syslog, другой – для ведения логов запросов веб-серверов; то есть, степень расширяемости логов обоих этих проектов была явно недостаточной для поставленных задач.

По результатам исследований для решения поставленных задач нами была разработана система трассировки и анализа трасс Aspectrace [8]. В данном разделе описывается её архитектура и предоставляемые ею возможности.

2.2. Структура трассы

Трасса состоит из последовательных сообщений.

Сообщение – минимальная единица информации в трассе. Каждое сообщение имеет заголовок и внутренние данные. Формат заголовка фиксирован: он содержит информацию об аспекте сообщения, его типе и канале. Возможные типы сообщений определяются аспектом, а синтаксическая структура внутренних данных сообщения определяется его аспектом и типом. В качестве базовой структуры внутренних данных сообщений выбраны атрибутированные деревья, на которые аспекты накладывают собственные синтаксические и семантические ограничения.

Канал представляет собой источник последовательных сообщений, гарантирующий на них линейный порядок. Например, разные каналы могут представлять собой источники сообщений, расположенные на разных машинах в сети или в разных нитях (threads) управления. Каждый канал имеет уникальный идентификатор, отличающийся от идентификаторов других каналов. В трассе могут содержаться вперемешку сообщения из разных каналов; при этом в общем случае относительный порядок сообщений, относящихся к разным каналам, недостоверен. В трассе также может содержаться информация, позволяющая упорядочивать сообщения из разных каналов, в том числе предназначенные специально для этой цели служебные сообщения.

Аспекты представляют собой различные срезы информации о целевой системе. Каждый аспект определяет собственную модель данных, допустимые виды сообщений, их синтаксические структуры и семантические ограничения на них. Также определены несколько видов служебных сообщений; они отнесены к особому служебному аспекту. С точки зрения программиста, использующего данную технологию, каждый аспект представлен подключаемым модулем, предоставляющим интерфейсы для работы с соответствующей моделью данных, сброса в трассу предусмотренных аспектом типов сообщений и для получения соответствующих сообщений при анализе трасс.

Различные компоненты целевой системы в ходе работы могут независимо друг от друга сбрасывать в трассу разнообразную информацию, относящуюся к различным аспектам её поведения. Особенно это существенно при тестировании, когда надо одновременно следить за работой и тестируемой системы (которая может содержать отладочный код, сбрасывающий внутреннюю информацию о своей работе, как изначально, так и в результате инструментирования, осуществлённого для целей тестирования), и тестовых компонентов (генераторов тестовых данных, тестовых оракулов, вспомогательных компонентов и так далее). Как уже отмечалось выше, по мере усложнения целевой системы добавляются новые виды информации, которые необходимо учитывать; это создаёт проблемы при любом фиксированном формате трассы, так как при расширении набора собираемой информации теряется совместимость по данным между целевыми и анализирующими приложениями. В технологии Aspectrace возможность гибко

добавлять в трассу новые виды информации обеспечивается с помощью расширяемого набора независимых аспектов. При возникновении необходимости собирать и анализировать информацию какого-либо нового вида, разработчики добавляют в целевую систему трассировку относящейся к соответствующим аспектам информации, используя соответствующие готовые библиотеки аспектов или разрабатывая новые. Параллельно с этим разрабатываются специализированные инструменты, анализирующие эту информацию, или добавляются соответствующие возможности к уже имеющимся инструментам анализа. Каждый инструмент, анализирующий трассу, нацелен на обработку определённых аспектов; при этом каждый такой инструмент выбирает из трассы только информацию от тех аспектов, которые он умеет обрабатывать, и которые нужны для построения запрошенного отчёта, игнорируя всё остальное.

Поскольку информация от разных аспектов сбрасывается в трассу и извлекается из неё при анализе независимо, процессы разработки создающих трассу приложений и инструментов для их анализа могут идти в значительной степени независимо, при этом совместимость между различными версиями трасс и инструментов их анализа сохраняется в обе стороны. Так, если целевая система добавляет трассировку новой информации и обгоняет инструменты анализа, то старые инструменты анализа просто игнорируют эту информацию и анализируют только то, что могут; если же инструменты анализа обгоняют целевую систему, то они просто не получают информацию соответствующего вида и генерируют отчёт только по тем данным, которые им предоставлены.

2.3. Компонентная архитектура

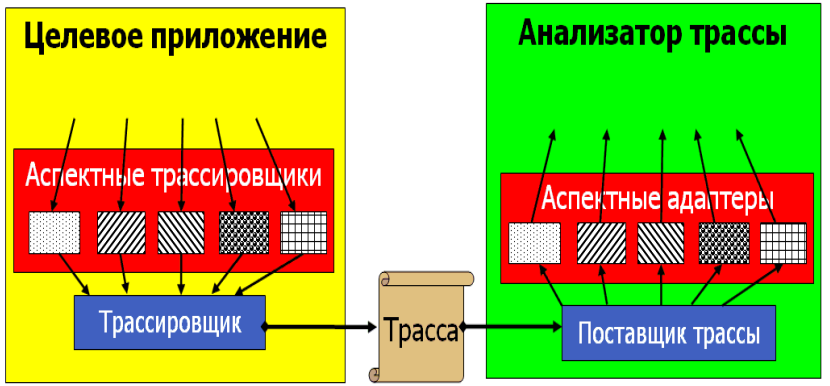

Рис. 1. Поток данных через компоненты.

На рис. 1 изображена структура компонентов системы трассировки и анализа трасс и способы её использования приложениями: как генерирующими трассы, так и анализирующими их.

Трассировщик и *поставщик трассы* – центральные инфраструктурные компоненты, поддерживающие расширяемый формат трассы и управляющие обработкой трасс. Они не поддерживают никаких моделей данных, обладающих самостоятельной ценностью для разработчика или аналитика, использующего трассы для мониторинга целевой системы; для сброса информации в трассу и выборки её оттуда приложения взаимодействуют с ними не напрямую, а через библиотеки аспектов.

Аспектные трассировщики и *аспектные адаптеры* входят в соответствующие библиотеки аспектов (на рисунке одинаковые аспекты обозначены одинаковыми текстурами) и подключаются приложениями по необходимости, независимо друг от друга. Аспектные трассировщики предоставляют приложениям, генерирующим трассы, интерфейсы для сброса сообщений в терминах модели данных своего аспекта; полученные данные преобразуются в универсальное промежуточное представление и передаются трасировщику. Аспектные адаптеры получают от поставщика трассы данные в промежуточном представлении, проверяют их на соответствие определённым в аспекте синтаксическим и семантическим ограничениям, преобразуют в модель данных аспекта и передают их анализирующим компонентам, которые для этого подписываются на получение от них соответствующих сообщений. В случае, если для какого-то аспекта не зарегистрировано ни одного подписчика, сообщения этого аспекта игнорируются на уровне поставщика трассы и не подвергаются дальнейшей обработке.

Для передачи оттрассированной информации от целевого приложения к анализатору трассы, а также для хранения трасс используются внешние представления трассы. Внешнее представление может быть, например:

- XML-файлом, хранящимся в файловой системе;
- Схемой базы данных под управлением внешней СУБД;
- Сетевым соединением, по которому передаются сообщения между брокерами объектных запросов (ORB) общего назначения с помощью имеющихся у них средств сериализации сложных объектов (COM, CORBA, Java/RMI и другие);
- Прямой связью между генератором и анализатором трассы, работающими в одном адресном пространстве, по которой ссылки в памяти на трассируемые сообщения передаются напрямую, без дополнительных преобразований (строго говоря, такое представление не является «внешним»);
- Любым другим каналом, допускающим передачу от целевой системы к анализатору трассы структурированных сообщений.

Различные внешние представления трассы поддерживаются соответствующими подключаемыми модулями, которые разработчики

системы могут подбирать в зависимости от особенностей окружения, в котором работают генераторы и анализаторы трассы.

3. Анализ трасс

В ходе работы приложения, генерирующего трассу, различные его компоненты сбрасывают информацию, относящуюся к различным аспектам его поведения. Компоненты генерирующей трассу системы могут разрабатываться и, соответственно, трассировать нужную им информацию независимо друг от друга, а могут иметь тесно связанную логику трассировки.

Получающаяся трасса похожа по структуре на медиаконтейнеры. Особо следует упомянуть контейнер Matroska («Матрёшка») [9] – открытый расширяемый формат контейнера, позволяющий добавление в медиафайл новых видов информации. В современных медиафайлах могут идти вперемешку блоки информации, описывающие, например, видео, звуковые дорожки, субтитры на разных языках, разбиение на разделы, синхронизационные метки и многое другое. Вся эта информация, представленная идущими последовательно разнородными блоками, с одной стороны, логически разделяется на идущие параллельно потоки (например, поток видео и поток звуковой дорожки), а с другой – как-то связана между этими потоками. Простейший механизм связывания информации между потоками – синхронизационные метки, но возможны и более сложные зависимости: например, звуковой поток и поток субтитров могут быть помечены одним и тем же языком, и проигрыватель может учитывать эту зависимость. Приложения, воспроизводящие информацию из медиаконтейнеров, используют соответствующие кодеки, чтобы извлекать и обрабатывать эти независимые потоки, после чего самостоятельно связывают их между собой и преобразуют всё вместе в понятный для человека вид (например, в видеоряд).

Примерно таков же принцип работы анализаторов трасс в архитектуре Aspectrace: они извлекают с помощью аспектных адаптеров нужные им поаспектные (а также поканальные) потоки сообщений, на анализ которых они нацелены, после чего анализируют их, находят связи между ними в соответствии с заложенной в эти анализаторы логикой и преобразуют всё вместе в понятный для человека вид.

Одни из этих связей могут быть определены структурой предметной области: например, сообщение в трассе об отправке сетевого пакета одним компонентом и сообщение об его получении другим должны содержать один и тот же идентификатор этого пакета. Другие связи могут быть заложены на этапе проектирования генерирующих трассу компонентов: например, если мы собираем информацию о покрытии исходного кода, то она будет коррелировать с информацией о вызовах методов, так как после вызова метода мы попадаем в соответствующий участок кода, и в этом случае имеет смысл сбрасывать в трассу сообщения о покрытии кода так, чтобы они

содержали указание на метод, в котором находится соответствующий код. Если нам известны связи этих двух видов, то в большинстве случаев разумно закладывать информацию о них в анализаторы трассы.

Третьи связи могут быть вообще не предусмотрены разработчиками приложений, генерирующих трассы, и именно их поиск будет целью трассировки и последующего анализа трасс: например, это могут быть корреляции между сообщениями об ошибках, сбрасываемыми в трассу одними компонентами целевой системы, и сообщениями о выполняемых действиях, сбрасываемыми другими компонентами.

3.1. Связывание плоских данных

Под «плоскими» здесь и далее мы понимаем данные, при анализе связей между которыми не учитывается время. Методы связывания данных на основе времени рассматриваются в следующем разделе.

Основные способы связывания и агрегирования плоских данных в целом известны нам из теории реляционных баз данных [10]. Для этих методик анализа неважно, являются ли данные однородными (относящимися к одному аспекту) или разнородными. Хотя каждый аспект предоставляет собственную модель данных, не зависящую от моделей данных других аспектов, использующее их приложение может строить над ними собственную модель, в которой присутствуют дополнительные межаспектные связи. Например, если мы оцениваем покрытие по вызовам тестируемых методов, то каждый вызов тестируемого метода (который может быть важен и для других видов анализа) также рассматривается как достижение соответствующего элемента покрытия; в этом случае при вызове метода в трассу сбрасываются (возможно, различными компонентами тестовой системы) два сообщения: сообщение о вызове и его параметрах в аспекте «вызовы методов» и сообщение о достижении элемента покрытия в аспекте «покрытия». Эти два сообщения содержат данные, которые позволяют при анализе связать их между собой; например, соответствующие методу сообщения о покрытии могут содержать указание на покрытие по вызовам методов и сигнатуру вызванного метода. В архитектуре Aspectrace нет таких жёстких связей между данными из разных аспектов, как, например, между соответствующими первичным и внешним ключами различных таблиц в реляционных БД, поэтому возможна полная изоляция данных из разных аспектов и независимый их анализ. Так, в данном примере генератор отчёта о покрытиях, ничего не знающий про методы, может, тем не менее, сгенерировать осмысленный отчёт о достигнутом покрытии. Аналогично, генератор отчёта о вызванных методах может сгенерировать осмысленный отчёт о них, игнорируя сведения о покрытиях. Однако анализатор трассы, понимающий данные обоих аспектов и знающий про эти связи, может пользоваться ими и анализировать более сложные межаспектные зависимости; в результате, например, отчёт о покрытии вызовов метода может, дополнительно к однообразной для всех отчётов о

покрытии информации, также ссылаться на дополнительную информацию об этом методе и его вызовах.

Кратко перечислим основные способы связывания плоской информации:

- Восстановление ссылок между данными сообщений (в том числе относящихся к разным аспектам). Например, описанный в примере выше в этом разделе пример со ссылками из данных аспекта «покрытие» на данные аспекта «вызовы методов», где ссылочным ключом служит сигнатура метода, размещённая в сообщении о достижении элемента покрытия так, что её могут извлечь соответствующие анализаторы трассы. Аналогом такого связывания в реляционной алгебре являются операции соединения [10].

- Простая агрегация. Например, генератор отчёта может собирать информацию о количестве вызовов указанного метода, или о времени, проведённом в его вызовах, или о наборе достигнутых и недостигнутых элементов описанной в той же трассе структуры покрытия.

- Связывание вида «класс–экземпляр». Например, в трассе может содержаться информация о вызовах метода, которые для данного связывания являются «экземплярами», а информация о самом методе является «классом». Такое связывание также является разновидностью агрегации, но имеет дополнительные важные свойства: во-первых, информация о классе может присутствовать в трассе, а может отсутствовать в ней в явном виде и восстанавливаться анализатором только на основе информации об экземплярах (по сути такое связывание аналогично запросам вида `SELECT … GROUP BY` языка SQL); во-вторых, при данном виде анализа в модели анализатора существуют и класс как целое, и отдельные его экземпляры, при этом класс и экземпляры могут ссылаться друг на друга.

3.2. Модельное время

Трасса содержит в себе не только плоские, но и темпоральные данные, которые также могут быть существенны при её анализе. Приложение, генерирующее трассу, сбрасывает в неё информацию в виде отдельных сообщений; информация из трассы поступает на вход анализатору также в виде последовательности сообщений. И отправка сообщения в трассу, и получение его оттуда анализатором являются атомарными одномоментными событиями, однако генерирующее эти сообщения приложение исходит из некоторой модели данных, сущности в которой могут иметь совсем другую структуру связей со временем и между собой, чем простая последовательность событий. Анализатор трассы, получая через трассу сообщения от анализируемого приложения, в большинстве случаев строит аналогичную

модель данных и пытается восстановить эти сущности (разумеется, если его вообще интересует информация такого вида).

В настоящем разделе мы рассматриваем темпоральные характеристики сущностей модели на примере аспекта, описывающего вызовы методов тестируемой системы. В модели данных этого аспекта могут присутствовать: методы, которые могут быть вызваны; их сигнатуры; вызовы и их параметры; возвращаемая информация и выбрасываемые исключения; стек вызовов; разнообразная дополнительная информация об этих методах.

Темпоральные шкалы каналов трассы соответствуют темпоральным шкалам потоков управления приложения, генерирующего трассу. Время, в котором работает анализатор трассы, для настоящего исследования несущественно, поэтому далее мы везде говорим о темпоральной шкале трассы, учитывая при необходимости, что может параллельно существовать несколько таких шкал, соответствующих разным каналам. Отметим также, что в трассе могут вообще отсутствовать какие-либо темпоральные метки; в этом случае в качестве таковых можно рассматривать, например, порядковые номера сообщений в канале. Такая темпоральная шкала, разумеется, не подходит для анализа производительности или временны**х** допусков на выполнение критичных действий, однако, обеспечивает анализатору линейную упорядоченность событий в рамках канала, позволяя таким образом анализировать взаимное расположение событий во времени, а также проверять условия темпоральной логики.

В результате исследований мы выделили следующие классы модельных сущностей в зависимости от их связи с модельным временем:

- Одномоментные события. К таким сущностям можно привязать единственную темпоральную метку (в качестве таковой обычно удобно использовать метку сообщения трассы, обозначающего наступление соответствующего события). Например, одномоментной сущностью может моделироваться событие вызова метода с возвратом, если продолжительность этого вызова и произошедшие внутри него события несущественны.

- Продолжительные события. Такие сущности имеют время начала и конца (в качестве которых обычно берутся темпоральные метки сообщений трассы, обозначающих соответственно начало и конец события), в результате к ним оказываются привязаны интервалы на темпоральной шкале, которые могут частично перекрываться, а также строго включать друг друга и темпоральные метки одномоментных событий. Например, продолжительными сущностями моделируются вызовы методов от момента собственно вызова до момента возврата (тем или иным образом), если в промежутке могут происходить другие вызовы, которые нас интересуют; в этом случае мы можем легко анализировать вложенность вызовов методов друг в друга.

- Глобальные. Такие сущности не имеют отношения к каким-либо моментам времени (несмотря на то, что конкретные сообщения, содержащие информацию о них, могут иметь темпоральные метки). Например, глобальной сущностью в модели является сам вызываемый метод; он может иметь сигнатуру, ссылку на исходный код и другие атрибуты, но все они существуют вне времени.

Отметим также, что существуют такие языки и платформы, в которых методы создаются и уничтожаются динамически, а также модели для них, в которой время существования методов важно – в этом случае в соответствующей модели метод перестаёт быть глобальной сущностью, а становится такой же продолжительной, как и его вызов. В общем случае темпоральная характеристика сущностей модели зависит от того, какие свойства моделируемой области учитываются в используемой при анализе модели.

3.3. Связывание данных по времени

Для анализа данных, полученных из одного канала трассы, наиболее широко применяются такие темпоральные характеристики событий трассы как последовательность событий и вложенность в продолжительные события одномоментных и других продолжительных событий. Поскольку темпоральная шкала не зависит от аспекта, возможно автоматическое построение связей между данными из разных аспектов, причём для этого не нужны компоненты анализатора, понимающие модели данных сразу от обоих аспектов, а достаточно двух независимых компонентов, каждый из которых нацелен на обработку данных от одного аспекта, и соответствующим образом настроенной связи между этими компонентами (подробнее механизмы такого связывания описаны в разделе «Генерация отчётов»).

Рассмотрим такое связывание на примере простой трассы. Допустим, у нас есть тестовые компоненты, сбрасывающие сообщения в аспектах «тест» (о выполняемых тестовых сценариях), «метод» (о запусках методов теста и их завершении), «покрытие» (о достижении элементов покрытия) и «ошибка»; а также тестируемая система, инструментированная (например, с помощью AspectJ [11]) для сбрасывания в аспекте «метод» сообщений о вызовах своих методов и возвратах из них. В таблице 1 приведён простой пример трассы, которая может получиться при работе такого теста. В колонке со скобками в заголовке обозначены точки начала и конца продолжительных событий.

№	()	Аспект: сообщение
1	<	Тест: Начало теста t1
2	<	метод: Вызов метода testf(1,2,3)
3		покрытие: Достигнут элемент покрытия «хорошие данные»
4	<	метод: Вызов метода f(4,5,6)
5	>	метод: Возврат из метода f() с возвращаемым значением «0»
6	>	метод: Возврат из метода testf() с возвращаемым значением «0»

7	>	тест: Конец теста t1
8	<	тест: Начало теста t2
9	<	метод: Вызов метода testf(7,8,9)
10		покрытие: Достигнут элемент покрытия «плохие данные»
11	<	метод: Вызов метода f(10,11,12)
12	>	метод: Возврат из метода f() с исключением «NullPointerException»
13		ошибка: Ошибка в аспекте «метод», текст = «null указатель»
14	>	метод: Возврат из метода testf() с возвращаемым значением «1»
15	>	тест: Конец теста t2

Таблица 1. Пример трассы.

В данном примере ни тестируемый компонент, ни встроенный в него код трассировки ничего не знают о тестах, которые над ними выполняются; тест также может состоять из независимых модулей, выполняющих тестовые сценарии, определяющих достижение интересующих тестировщиков тестовых ситуаций, выполняющих другие действия и сбрасывающих обо всех этих событиях сообщения в трассу. Однако за счёт вложенности событий трассы друг в друга мы можем при её анализе легко соотнести обнаруженные ошибки и достигнутые элементы покрытия с тестом, в ходе выполнения которого они произошли, а также с вызовами тестовых и тестируемых методов и параметрами этих вызовов.

Описанные в разделе 3.1 методы связывания данных действуют в пределах одного канала, когда из трассы естественным путём извлекается информация о последовательности событий. В случае наличия нескольких каналов зависимости между данными могут быть не столь однозначны. Не существует общих способов восстановления последовательностей событий в таких целевых системах, пригодных для всех случаев; однако, существуют широко используемые (в частности, в технологиях разработки и анализа телекоммуникационных систем) и пригодные для достаточно широкого класса задач способы межканального связывания данных. Поскольку данная статья посвящена работе с гетерогенными данными, а не вопросам темпорального анализа, позволяющего связывать между собой однородные данные в различных временны**х** шкалах, мы не будем рассматривать здесь методы темпорального анализа более подробно, а ограничимся перечислением простейших способов такого связывания:

- Связывание по общему таймеру. Например, такой таймер может быть доступен различным процессам, выполняющимся на одной машине.
- Связывание по общим синхронизационным меткам. Например, если несколько компонентов в сети совместно выполнили протокол синхронизации (например, MPI Barrier [12]), и сбросили в трассу соответствующие сообщения, то мы можем с уверенностью считать, что все события, сообщения о которых находятся в любом из этих

каналов канале до синхронизации, произошли раньше, чем все события, произошедшие в любом из этих каналов после неё.

- Связь по данным сообщений. Во многих случаях возможно уникально идентифицировать сообщения, передаваемые в распределённой системе, и таким образом связывать события передачи сообщений в одних каналах трассы и события их получения в других каналах; в частности, события передачи сообщений заведомо происходят раньше событий их приёма.

4. Генерация отчётов

4.1. Задачи генератора отчётов и требования к нему

Обрабатывая трассу, анализатор строит некоторую модель извлечённых из неё данных. При этом он может не только восстанавливать те модели данных, которыми пользовалось создавшее трассу приложение, но и синтезировать на их основе свои собственные; в предыдущем разделе перечислены некоторые способы такого синтеза данных, в том числе относящихся к различным предметным областям и различным временны**м** шкалам. Поскольку одна и та же задача извлечения определённых данных, их анализа, преобразования и связывания между собой определённым способом может вставать при разработке разнообразных отчётов, возникает потребность в инфраструктуре, позволяющей реализовывать решение таких аналитических задач с помощью переиспользуемых компонентов, а затем с невысокими накладными расходами собирать из них готовые генераторы отчётов, в которых построенные этими компонентами модели данных автоматически связывались бы между собой.

После того как данные извлечены, проанализированы и связаны, их необходимо представить в виде, понятном разработчику или аналитику целевой системы – построить отчёт. Отчёт может быть представлен в виде графиков, таблиц, гипертекстовых страниц, изображающей поведение анализируемой системы анимации и во многих других формах. В любом случае, нужна инфраструктура, поддерживающая работу с выбранной формой представления данных. Важным требованием к такой инфраструктуре является возможность автоматически переводить связи (возможно, разных типов) между элементами модели данных в связи между соответствующими элементами представления.

Ниже описана архитектура генератора отчётов, позволяющая решать обе поставленные задачи. Для примеров используется генератор HTML-отчётов, но описанные принципы применимы и к другим формам представления данных.

4.2. Компонентная архитектура генератора отчётов

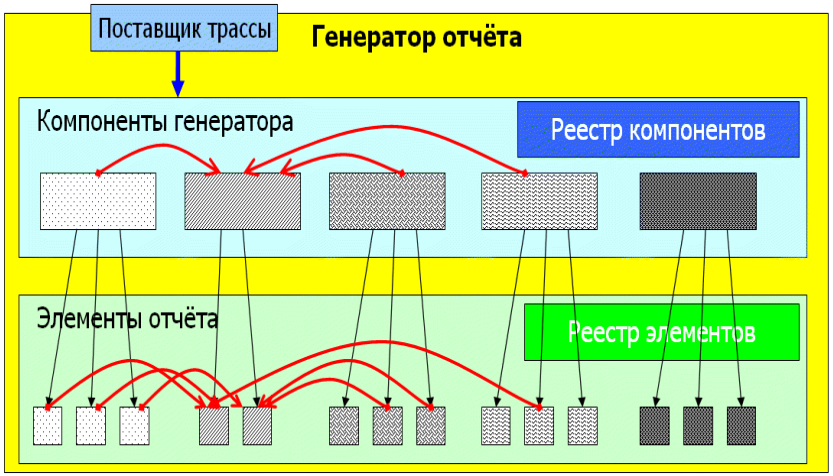

Рис. 2. Компоненты генератора и элементы отчёта.

На рисунке 2 изображена компонентная структура генератора отчётов. Ядро генератора отчётов предоставляет разработчику отчёта интерфейс для подключения к нему компонентов генератора и настройки связей определённых в нём видов между ними, а подключенным компонентам – интерфейс для добавления в отчёт элементов.

Компоненты генератора (дальше называемые также просто «компонентами») являются анализаторами трассы: они подсоединяются к общему поставщику трассы, получают из него сообщения от нужных им аспектов и анализируют их. На основе полученных данных они строят элементы отчёта соответствующих генератору видов (например, в случае генератора HTML-отчётов это будут целые HTML-страницы или их части) и добавляют их в отчёт. Компоненты могут предоставлять интерфейсы для установления связей с другими компонентами, а также для запроса контролируемых ими элементов отчёта.

Генератор отчётов ведёт реестры своих компонентов и элементов отчёта. Разные реестры могут поддерживать разные пространства имён и разные способы поиска по этим именам и связывания хранящихся в них сущностей (компонентов генератора или извлечённой ими информации). Компоненты генератора могут вести собственные реестры, хранящие информацию о созданных ими элементах отчёта, и предоставлять другим компонентам возможность осуществлять запросы к ним напрямую или подсоединять их к реестрам-посредникам ядра генератора, диспетчеризующим обращённые к ним запросы в соответствии со своей конфигурацией.

Компоненты генератора могут также регистрироваться в существующих реестрах компонентов, находить друг друга через них и запрашивать у найденных компонентов (формально говоря, у реестров, которые те ведут) нужные им элементы отчёта. Компоненты, между которыми установлены связи, с их помощью устанавливают соответствующие связи между своими элементами отчёта; на рисунке 2 такие связи обозначены дуговыми стрелками. Также ядро генератора отчётов может самостоятельно связывать между собой включённые в отчёт элементы, в соответствии со своими настройками и прикреплённой к этим элементам мета-информацией.

Некоторые способы связывания элементов отчёта описаны концептуально и на примерах ниже; подробнее ознакомиться с конкретными примерами реализации этих концепций можно на сайте проекта Aspectrace [8].

Для конструирования генератора отчёта необходимо:

1. Выбрать нужный вид отчёта (например, отчёт в виде набора HTML-документов) и поддерживающее его ядро генератора отчётов (например, входящее в поставку системы).
2. Подобрать из библиотеки готовые или разработать нужные компоненты генератора отчётов.
3. Собрать из выбранных компонентов конфигурацию генератора:
   a. Настроить получение компонентами генератора информации из нужных аспектов от поставщиков трассы.
   b. Настроить эти компоненты.
   c. Настроить нужные связи компонентов генератора отчётов друг с другом и с общими реестрами.
   d. Настроить в ядре генератора его собственные (не зависящие от подключённых компонентов) способы автоматического связывания элементов отчёта.

Когда генератор отчётов сконструирован, остаётся только указать ему входные трассы, запустить фазу анализа, а после её завершения – пост-обработку, если эта фаза предусмотрена типом генератора. В результате на выходе мы получаем экземпляр отчёта, описывающий конкретные полученные данные. Конструирование генератора отчёта можно выполнить один раз, после чего подавать на вход собранной конфигурации генератора различные входные трассы и получать на выходе соответствующие им отчёты.

4.3. Связывание элементов отчёта

В этом разделе рассматриваются вопросы, относящиеся к связыванию данных, уже извлечённых разными компонентами генератора и преобразованных в модель отчёта соответствующего вида. Вопросы связывания информации

внутри одного компонента рассмотрены выше, в разделе 3; в случае различных компонентов все эти принципы связывания также применимы, однако возникают дополнительные технические вопросы интероперабельности между компонентами.

Для связывания элементов отчёта между собой необходимо определить:

1. Как одни компоненты будут находить нужные им элементы отчёта, созданные другими компонентами. К каким реестрам они будут для этого обращаться и каким образом будут извлекать из входного потока информации нужные для поиска ключи.

2. Каким образом будут реализовываться связи между соответствующими элементами отчёта.

Существует большое разнообразие решений этих двух вопросов. В частности, элементы отчёта могут искаться в разных реестрах по ключам, относящимся к разным пространствам имён и получаемым разными способами из трассы или от других компонентов; эти реестры могут статически содержать искомые элементы отчёта или переадресовывать поисковые запросы соответствующим компонентам; элементы могут существовать в реестре независимо от запросов, создаваться в ответ на запрос или генерировать при запросе элемент-посредник. Способы связывания элементов тоже могут быть разнообразны; например, установить связь между двумя HTML-страницами можно с помощью проставления гиперссылок, включения одной страницей внутрь себя другой (так что та становится её частью, и все гиперссылки на вторую страницу перенаправляются на соответствующую метку внутри первой), её копии или некого производного текста, а также включения одной страницей другой в качестве фрейма. Другие виды отчётов могут поддерживать другие представления данных и связей между ними; при этом если соответствующие элементы отчёта поддерживают общие интерфейсы для установления связей между собой, то управляющие ими компоненты генератора отчётов могут устанавливать связи между своими элементами отчёта и элементами, созданными другими компонентами, не имея никакой дополнительной информации друг о друге.

Выбор конкретных решений не регламентируется технологией и зависит только от соглашений, принятых при разработке компонентов генератора, а также при сборке из них конкретного генератора отчётов; при этом в рамках одного генератора возможно одновременное использование разнообразных соглашений.

Наиболее простым и естественным способом связывания данных между компонентами является разметка участков трассы в сочетании с прослеживанием элементов отчёта до сообщений анализируемой трассы, на основе которых они созданы. Непрерывный участок трассы, помеченный некоторой меткой, определяет продолжительное событие (подробнее о продолжительных событиях см. в разделах 3.2–3.3), которому может

соответствовать некоторый элемент отчёта; с помощью таких меток можно естественным и технически несложным способом устанавливать связи между этим элементом отчёта и другими, которым соответствуют другие события в трассе, как вложенные в это событие, так и содержащие его.

Разметка трассы может быть явной и неявной. При явной разметке для любого сообщения трассы можно узнать набор меток, которые действуют в данной точке трассы (поскольку хранение системой анализа трасс всех сообщений и всех меток для них было бы слишком неэффективно, обработка сообщений идёт в потоковом режиме; поэтому запрос действующих на сообщение трассы пометок можно делать только в момент обработки этого сообщения), и запросить (сразу и/или после сбора всей информации из трассы, в зависимости от конкретного используемого механизма разметки, особенностей запрашиваемой информации и соглашений между компонентами генератора) в соответствующих реестрах элементы отчёта, соответствующие этим меткам.

В ядро подсистем трассировки и анализа трасс входит один механизм такого типа, называемый механизмом тэгов. Тэг представляет собой строку, квалифицированную именем аспекта. Для тэга определены операции установки и снятия в заданном канале; всё сообщения в соответствующем канале от точки установки тэга до точки его снятия считаются помеченными им. Компоненты целевого приложения могут устанавливать и снимать тэги, сбрасывая соответствующие сообщения в трассу; другие тэги устанавливаются и снимаются компонентами, анализирующими трассу, в ходе анализа. При этом первые тэги присутствуют в трассе, независимо от её формы внешнего представления, а вторые существуют только в процессе работы анализирующего трассу приложения, содержащего ставящий этот тэг компонент, но всем компонентам анализатора видны тэги обоих видов. Компонент генератора отчётов, поставивший на участке трассы собственный тэг, регистрирует в соответствующем реестре элемент отчёта, соответствующий ему. Также компоненты генератора отчёта могут на основании известной им информации о том, каким образом расставлялись тэги, пришедшие из входной трассы, отображать их с помощью соответствующего реестра в свои элементы отчёта.

Например, в трассе, приведённой в таблице 1, будут присутствовать следующие тэги:

- «тест:t1», действующий на сообщения 1–7
- «метод:testf», действующий на сообщения 2–6
- «метод:f», действующий на сообщения 4–5
- «тест:t2», действующий на сообщения 8–15
- «метод:testf», действующий на сообщения 9–14
- «метод:f», действующий на сообщения 11–12

Тэги от аспекта «тест» уникально идентифицируют тест, и компонент генератора, обрабатывающий сообщения от этого аспекта, может отображать их в элементы отчёта, описывающие соответствующие тесты, позволяя таким образом связывать с ними, например, элементы отчёта, соответствующие сообщениям №№3 и 10 (достижение элементов покрытия), а также элемент, соответствующий сообщению №13 (обнаружение ошибки), с соответствующими тестами.

Однако, как мы видим, тэги вида «метод:testf» и «метод:f» встречаются более одного раза, а значит, их однозначное отображение в элементы отчёта, соответствующие событиям вызовов методов, невозможно. Поэтому библиотека аспекта «метод» устанавливает также дополнительные тэги:

- «метод:testf:1», действующий на сообщения 2–6
- «метод:f:2», действующий на сообщения 4–5
- «метод:testf:3», действующий на сообщения 9–14
- «метод:f:4», действующий на сообщения 11–12

Уникальные тэги аналогичного вида могут устанавливать анализирующие трассу компоненты, обеспечивая таким образом уникальность ключей для поиска элементов отчёта, соответствующих вызовам методов, в рамках не только одной трассы, но и всего набора трасс, поступивших на вход генератору отчёта.

Искать в реестре с помощью тэгов элементы отчёта и связывать их между собой могут не только компоненты генератора отчётов, но и само ядро генератора по указанным ему правилам. Правила эти могут иметь, например, такой вид: «для всех элементов отчёта, помеченных как относящиеся к аспекту А и помеченных тэгами из аспекта Б, для каждого такого тэга искать в реестре элемент отчёта, соответствующий ему, и если таковой найден – устанавливать указанным образом связь между первым и вторым элементом». Связывание такого рода выполняется в фазе пост-обработки; для использования этой возможности, элементы отчёта, относящиеся к аспекту А, должны хранить тэговые метки, которыми были помечены сообщения трассы, на основе которых они созданы.

При неявной разметке трассы явных меток нет, а связывание проводится по динамическому контексту анализа входной трассы. Компоненты, анализирующие трассу, могут при получении очередного сообщения трассы запросить у заданных конфигурацией генератора реестров элементы отчёта, ассоциированные с текущей точкой трассы (такие запросы, в отличие от запросов явной разметки, не могут быть отложены до завершения обработки трассы). Поскольку все компоненты генератора отчётов подсоединены к общему источнику трассы, а сообщения от него обрабатываются строго последовательно, все компоненты имеют общий контекст обработки трассы; при этом, однако, разные компоненты могут получать сообщения от разных

аспектов и изменять набор элементов отчёта, которые они ассоциируют с текущей точкой трассы, в ответ на разные сообщения.

Например, при обработке трассы из таблицы 1 компонент, обрабатывающий сообщения от аспекта «тест» может хранить ссылку на элемент отчёта, соответствующий текущему (во времени обрабатываемой трассы) выполняемому тесту: при обработке сообщения №1 (начало теста t1) эта ссылка будет установлена на элемент отчёта, соответствующий тесту t1, при обработке сообщения №7 (конец теста t1) – сброшена, при обработке сообщения №8 установлена на элемент отчёта, соответствующий тесту t2, а при обработке сообщения №15 – опять сброшена. Соответствующий реестр, поддерживаемый этим компонентом, принимает запросы (отметим, что ключ для запросов может быть пустым), в ответ на которые возвращает ссылку на текущий тест. Компонент, обрабатывающий сообщения от аспекта «ошибка», получив сообщение №13 (ошибка), обращается к этому реестру и получает ссылку на элемент отчёта, соответствующий тесту t2, после чего устанавливает связь между ним и своим элементом отчёта, соответствующим этой ошибке.

Второй после разметки трассы основной способ связывания элементов отчёта – прямой их поиск через реестры по ключам, извлекаемым из сообщений трассы. Способы извлечения ключей зависят от известной информации о зависимостях между сообщениями, сбрасываемыми в трассу различными компонентами целевой системы; пример такого связывания приведён в разделе 3.1. Сам по себе поиск по реестрам и общие принципы работы с реестрами также уже описаны: выше в данном разделе, а также в разделе 4.2, поэтому мы не будем расписывать здесь этот механизм заново.

5. Заключение

При анализе работы сложных систем, особенно состоящих из множества взаимодействующих компонентов, в том числе при их тестировании, возникает необходимость собирать в виде трасс большое количество весьма разнообразной (гетерогенной) информации, а потом анализировать её и представлять в виде понятных разработчикам или заказчикам отчётов. В ходе разработки целевой системы набор видов трассируемой информации может часто меняться, обычно в сторону расширения. Разнообразие и частое изменение видов информации в трассе создаёт проблемы при любом фиксированном формате трассы, так как при расширении набора собираемой информации теряется совместимость по данным между целевыми и анализирующими приложениями.

Поскольку сбор одинаковых видов информации и схожие виды её анализа могут использоваться при разработке разнообразных систем, возникает потребность иметь готовые анализирующие модули, которые можно было бы переиспользовать в других проектах и в других отчётах.

В статье рассмотрены задачи, возникающие при сохранении гетерогенной информации и её анализе, и описана архитектура системы, поддерживающей создание, хранение и анализ гетерогенных трасс. Рассмотрены разнообразные методы анализа и связывания гетерогенных данных, как с использованием темпоральной информации, так и без него. Описана архитектура генератора отчётов, позволяющего собирать генераторы отчётов из готовых независимых аналитических модулей, автоматически связывать между собой результаты проведённого ими анализа и представлять извлечённые данные в виде единого отчёта, понятного человеку.

Литература

[1] Постоянная ссылка на версию статьи:
http://ru.wikipedia.org/wiki/Трассировка_(программирование)?oldid=20486892
[2] Постоянная ссылка на версию статьи:
http://en.wikipedia.org/wiki/Tracing_(software)?oldid=305164770
[3] Ф. Крачтен. Введение в Rational Unified Process // Вильямс, 2002.
[4] К. Бек. Экстремальное программирование // Питер, 2002
[5] A. Cockburn. Agile Software Development: The Cooperative Game (2nd Edition) // Addison-Wesley Professional, 2006
[6] SystemTap homepage
http://sourceware.org/systemtap/
[7] eXtensible Logfile Format (XLF) 1.9.2 Specification
http://www.graphicaldynamics.com/xlf/xlf-spec-1.9.2.html
[8] Сайт проекта Aspectrace
http://forge.ispras.ru/projects/show/aspectrace
[9] Matroska media container homepage
http://www.matroska.org/index.html
[10] С.Д. Кузнецов. Основы баз данных // Бином, 2007
http://www.citforum.ru/database/osbd/contents.shtml
[11] The AspectJ Project homepage
http://www.eclipse.org/aspectj/
[12] Антонов А.С. Параллельное программирование с использованием технологии MPI: Учебное пособие // Издательство Московского Университета, 2004
http://parallel.ru/tech/tech_dev/MPI/