Автоматизированное создание тест-кейсов для тестирования соединений и протоколов

Сергей Павлов
Exactpro Systems
sergey.pavlov@exactprosystems.com

Андрей Соловьев
Exactpro Systems
andrey.soloviev@exactprosystems.com

Александр Бормотин
Exactpro Systems
alexander.bormotin@exactprosystems.com

Анна Торопова
Exactpro Systems
anna.toropova@exactprosystems.com

Иосиф Иткин
Exactpro Systems
iosif.itkin@exactpro.com

***Аннотация*—Современные брокерские, биржевые и клиринговые платформы неотделимы от финансовых протоколов, обеспечивающих интеграцию подсистем и подсоединение пользовательских систем. Верификация корректной работы протоколов и подключений является неотъемлемой частью тестирования любой реализации программного обеспечения для таких платформ. В данной статье рассмотрен комплексный подход к автоматизированной генерации тест-кейсов для тестирования различных финансовых протоколов. Выявлены требования к документации, необходимые для реализации данного подхода, а именно к способам описания словарей сообщений и способам описания состояний и переходов между ними. Рассмотрена реализация стандартного протокола распространения данных о котировках - ITCH. Предложены способы автоматизированной генерации тест-кейсов для его динамической верификации.**

***Ключевые слова*—протокол, тестирование протоколов, protocol testing, connectivity testing, XML, XSD, workflow, конечные автоматы, словарь, dictionary, Finite State Machine, FSM, UML, SCXML, инструменты тестирования, testing tools**

I. Введение

Тестирование программного обеспечения является неотъемлемой частью цикла разработки, существует множество подходов и способов организации тестирования в частности, и цикла разработки в целом. С усложнением программного обеспечения ресурсы, необходимые для его тестирования, также увеличиваются. В особенности, это применимо к современным программным комплексам, обеспечивающим работу финансовых институтов, таких как брокеры, биржи, центральные депозитарии и клиринговые дома, которые являются сложными распределенными системами.

Одним из способов уменьшения затрат на обеспечение качества программного обеспечения является автоматизация процесса тестирования. Кроме того, автоматизация позволяет улучшить качество тестирования, то есть уменьшить количество ошибок, возникающих вследствие человеческого фактора. Существует множество различных инструментов, позволяющих автоматизировать выполнение уже готовых тест-кейсов. Однако, в большинстве случаев написание сценариев тестирования и разработка тест-кейсов остаются полностью ручными процессами.

Необходимо заметить, что финансовые платформы не только являются сложными системами, но и продолжают постоянно развиваться. Следовательно, с добавлением функций тестируемой системы приходится создавать новые тесты. Логичным выводом из этого следует, что стоит попытаться автоматизировать процесс создания тест-кейсов. Но перед этим необходимо принять во внимание два аспекта:

- Будут ли ресурсы, затраченные на создание инструмента и автоматизированную генерацию тест-кейсов, меньше ресурсов, необходимых для их ручного создания;
- Возможность использования созданного инструмента для генерации тестов для других областей или систем.

Тестирование протоколов и соединений являются областями, которые занимают много времени, если тесты создаются вручную. В то же время, принципы, по которым создаётся набор тестов, достаточно просты и применимы к большинству тестируемых систем. Более того, существуют широко распространенные протоколы, такие как FIX и SWIFT[1,2], поддерживаемые и используемые большинством организаций в финансовой индустрии. Поэтому инструмент для тестирования, создающий тест-кейсы можно будет использовать для большинства существующих систем. Именно здесь можно применить автоматизированную генерацию тест-кейсов.

II. Определения

A. Определения и типы протоколов

Современные торговые платформы, а так же системы проведения расчетов и клиринга обслуживают большое количество пользователей по различным протоколам.

Протокол передачи данных — это набор правил, позволяющих двум или более объектам осуществлять связь между собой. Правила определяют синтаксис, семантику, способ передачи сообщений и методы обработки ошибок при взаимодействии объектов. Как правило, в научной литературе описывается несколько широко известных парадигм классификаций протоколов. Например, модель OSI[13] состоит из семи упорядоченных уровней, а стек протоколов TCP/IP[17] состоит из четырех уровней. Финансовые протоколы, в большинстве своем, можно классифицировать как относящиеся к прикладному уровню. Однако, популярные классификации имеют широкий охват, и требуют дополнительной структуризации по определенным признакам. Наиболее часто прикладные протоколы дополнительно разделяют по типу представления информации на две группы: текстовые (символьные) и бинарные.

При текстовом представлении информации данные передаются в виде текста (строк), описанных как правило в виде JSON-строк, XML-сообщений (FIXML, SOAP) или тэгерированых сообщений (FIX, SWIFT MT). Несмотря на то, что сообщения в таком формате проще отлаживать и легче воспринимать, они имеют большой процент избыточной информации за счет сравнительно большого объема оболочки.

Бинарные протоколы, в отличие от текстовых, могут использовать различные типы данных (строковые, числовые, битовые поля) для представления информации. Поля, а следовательно и сообщения, могут иметь фиксированную или переменную длину.

Отметим, что возможна ситуация, когда в протоколе используются оба представления.

B. Технологический процесс (англ. - workflow)

Под технологическим процессом в данной статье подразумевается совокупность всех возможных состояний, включая начальное и конечное состояния, если таковые имеются, и событий, происходящих внутри рассматриваемой системы или ее части, а также связей между ними.

C. Спецификация протокола

Спецификацией протокола в этой статье будем называть набор требований к входящим и исходящим сообщениям. Спецификации протоколов так же могут ссылаться на технические стандарты.

III. Особенности и цели тестирования протоколов

A. Тестирование соединения

Большое количество одноадресных протоколов (англ. unicast), к которым, как правило, относятся и финансовые, не позволяет инициировать обмен прикладными сообщениями сразу после установки соединения на транспортном уровне. Перед началом обмена прикладными сообщениями клиент и сервер должны обменяться административными сообщениями, которые кроме установки соединения, осуществляют поддержку соединения, синхронизацию между клиентом и сервером, восстановление сообщений, закрытие соединения и другие функции поддерживаемые протоколом. Прикладные же сообщения используются непосредственно для обмена финансовой информацией. В отличие от прикладных сообщений, которые существенно отличаются в зависимости от конкретной системы, административные сообщения унифицированы. Например, сессионный уровень (административные сообщения) FIX протокола был выделен в отдельный подпротокол FIXT - FIX Transport[1]. В него входят всего 7 сообщений: Logon, Heartbeat, Test Request, Resend Request, Reject, Sequence Reset, Logout.

Тестирование соединения (англ. connectivity testing) относится к функциональному тестированию. В нем проверяются вышеназванные аспекты работы системы. В него входят как позитивные сценарии - например, Logon с правильной комбинацией имени пользователя и пароля, так и негативные, - например, Logon с неправильным паролем для данного имени пользователя.

Как правило, большая часть тестирования протокола происходит после позитивного тестирования подключения в связи с тем, что тестирование протоколов, чаще всего, невозможно без установленного соединения.

B. Тестирование протоколов

Под тестированием протоколов, как правило, подразумевается тестирование коммуникации двух или более систем (например, взаимодействие между системами брокера и биржи) с учетом специфики конкретной реализации протокола.

Тестирование протоколов условно можно разделить на две категории: негативное и позитивное тестирование.

1) Позитивное тестирование протоколов

Позитивное тестирование представляет собой набор тестов соответствующих штатному поведению системы. Если рассматривать тестирование протоколов, то позитивная часть тестирования, в большинстве случаев, представляет собой комбинацию всех возможных корректных сообщений и ответов на них. Однако, такой подход порождает большое количество тестов и некоторая их часть будет избыточна. При оптимизации процесса тестирования логично использовать широко распространенные подходы, такие как классы

эквивалентности, для определения достаточного набора тестов.

Важно подчеркнуть, что сценарии тестирования, различные с точки зрения общей логики системы (функциональные тесты), могут содержать одни и те же входные и выходные сообщения, то есть быть идентичными с точки зрения тестирования протоколов.

В общем случае представляется, что в любой системе реализация протокола должна быть необходима и достаточна для взаимодействия между клиентом и сервером. Таким образом, при покрытии всей логики системы функциональными тестами, автоматически проверяется позитивное тестирование протоколов, и его отдельная проверка становится ненужной.

Когда же нужно позитивное тестирование протоколов? Мы полагаем, что прежде всего оно необходимо в начале тестирования системы или новой функциональности кем бы то ни было: внутренняя или внешняя команды обеспечения качества программного обеспечения, конечные пользователи. Тестирование протоколов выявляет дефекты на ранних стадиях и уменьшает цену ошибки. Кроме того, оно также является частью тестирования совместимости клиентов.

2) Негативное тестирование протоколов

Негативное тестирование включает себя тесты, которые соответствуют нештатному поведению системы - различные ошибки, крайние состояния, некорректный ввод. Негативное тестирование протоколов подразумевает под собой посыл в систему некорректных сообщений, например, относительно словаря или с неправильным порядком сообщений. Целью таких тестов, как правило, является проверка устойчивости системы к различным входным данным, запись всех событий связанных с этими данными и проверка обработки граничных состояний.

Негативное тестирование протоколов обычно не осуществляется конечными пользователями системы. В особенности это касается пользователей с сертифицированными системами, которые должны посылать только правильные сообщения. Оно часто имеет меньший приоритет для тестировщиков, которые концентрируются на сценариях более вероятных в реальной промышленной системе. Однако возможные последствия неправильной реакции на некорректные сообщения могут быть гораздо серьезнее, чем ошибки при обработке правильного сообщения. Наш опыт показывает, что возможны ситуации, когда при получении неправильных сообщений появляются критические ошибки в системе, влияющие на ее дальнейшую работоспособность.

Разумеется, в большинстве случаев способов послать неправильное сообщение гораздо больше, чем послать корректное сообщение. Поэтому в негативном тестировании протоколов, даже больше чем в позитивном, важна оптимизация набора тестов.

Стоит отметить, что в отличие от позитивного тестирования, негативное должно быть включено в библиотеку регрессионных тестов из-за того, что эти тесты не проверяются другими сценариями.

Подготовка и проведение тестирования протоколов часто отнимает много времени и ресурсов. Но так как принципы, по которым создаётся набор тестов, достаточно просты и применимы к большинству тестируемых систем, именно здесь можно применить автоматизированную генерацию тест-кейсов.

IV. Требования и подходы к описанию словаря и технологического процесса

A. Требования к описанию спецификации протокола

При реализации протокольного соединения между компонентами клиентского и серверного приложений необходимо, чтобы все элементы системы могли четко и однозначно интерпретировать получаемые и отправляемые сообщения. Для этого необходимо описать словарь сообщений.

В настоящее время существует довольно много способов для описания словарей. Спецификация в текстовом формате является наиболее распространенным вариантом. Обычно такой документ поставляется вместе с системой или находится в свободном доступе. В файле описаны типы данных, формат протокола передачи данных, а так же структура сообщений. Пример описания сообщения UnitHeader протокола MITCH[11] можно увидеть в Таблице I.

Таблица I. Пример описания словаря в текстовой спецификации

Field	Offset	Length	Type	Description
Length	0	2	UInt16	Length of the message block including the header and all payload messages.
Message Count	2	1	UInt8	Number of payload messages that will follow the header.
Market Data Group	3	1	Byte	Identity of the market data group the payload messages relate to.
Sequence Number	4	4	UInt32	Sequence number of the first payload message.
Payload	5	Variable	-	One or more payload messages.

Текстовый формат описания словаря позволяет емко и точно описать все детали, но, как правило, имеет свободный формат и сложен для машинной обработки, для которой требуется описание в более формальном виде. Для этого существует ряд языков с простым синтаксисом, удобным для создания и обработки документов, как программами так и людьми. К таким языкам можно отнести XML, XSD, JSON.

В некоторых случаях вместе с системой поставляется описание протокола в одном из вышеперечисленных форматов, но чаще всего предоставляются только текстовые описания.

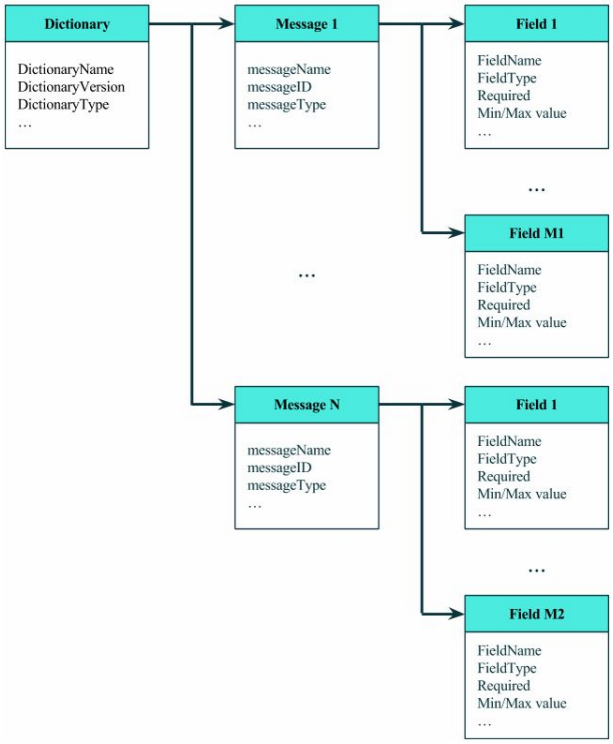

Рис. 1. Схематичное изображение структуры словаря

Несмотря на существование множества различных финансовых протоколов, можно выделить общую структуру словаря (см. Рис. 1) для любого из них:

- Название и атрибуты протокола;
- Список всех сообщений с их атрибутами;
- По каждому сообщению - список полей с их атрибутами.

Наиболее популярным способом описания протокола является язык XML. На верхнем уровне XML словаря описывается структура dictionary. Атрибутами этой структуры являются имя протокола, версия протокола, тип протокола.

Исходя из того, что протокол состоит из набора различных сообщений, структура dictionary включает в себя набор структур message. Атрибутами структуры message являются и идентификатор сообщения. В свою очередь, структура message состоит из набора структур field. Для каждой структуры field определены атрибуты FieldName и FieldType. Каждая структура field имеет набор определенных параметров, например длина поля.

Набор параметров поля варьируется и зависит от конкретной реализации, например существуют поля со сложной структурой, такие как повторяющиеся группы в протоколе FIX.

Описание словаря в формате XML имеет ряд недостатков. В частности, с помощью XML невозможно описать сложные структуры данных и шаблоны данных. Для описания сообщений, в которых применяются сложные поля, можно использовать язык XSD.

Язык XSD, в отличие от XML позволяет описать:

1. Сложный формат строки, используя паттерны, основанные на регулярных выражениях. Например, паттерн:

```
<xs:restriction base="xs:string">
<xs:pattern value="[A-Z]{6,6}[A-Z2-9][A-NP-Z0-9]([A-Z0-9]{3,3}){0,1}"/>
</xs:restriction>
```

2. Поле, которое представлено одним из нескольких вариантов. Например выражение:

```
<xs:complexType name="PartyIdentification36Choice">
<xs:sequence>
<xs:choice>
<xs:element name="AnyBIC" type="AnyBICIdentifier"/>
<xs:element name="PrtryId" type="GenericId19"/>
</xs:choice>
</xs:sequence>
</xs:complexType>
```

означает, что поле “PartyIdentification36Choice” может быть заполнено или значением типа “AnyBICIdentifier”, или значением типа “GenericId19”.

B. Технологический процесс

В отличие от описания словарей, на данный момент мы не видим примеров описания технологического процесса, заданного явным образом для протоколов, используемых торговых и расчетно-клиринговых системах. Чаще всего он описывается неявным образом документами в формате .pdf или .doc, в которых также описываются словарь и сопутствующие бизнес-процессы. Описание может быть представлено в виде комбинации текста, таблиц и различных диаграмм[3, 11].

В [4] отмечается, что типичная реализация FIX протокола имеет технологический процесс, часть которого может быть описана текстом, например:

- В момент времени t система пассивно ждет подключения со стороны клиента;
- В момент времени t2 клиент посылает сообщение Logon;
- В момент времени t3 система получает сообщение Logon и посылает ответное сообщение Logon;
- В момент времени t4 клиент посылает сообщение Heartbeat;
- В момент времени t5 система посылает сообщение;

Вышесказанное также может быть представлено в графическом виде (см. Рис.2)

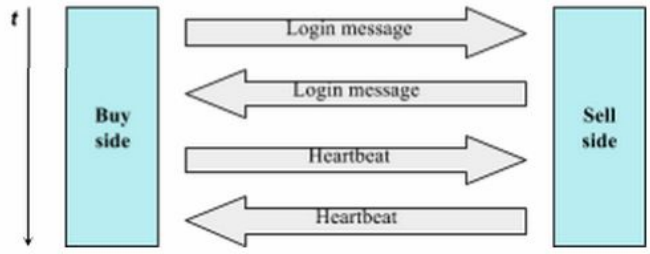

Рис. 2. Графическое представление части технологического процесса протокола FIX

Здесь шкала времени направлена вниз, то есть сообщение, расположенное выше, послано раньше чем сообщение, расположенное ниже.

В обоих случаях представление не является ни точным, ни полным. Очевидная неточность - это последовательность событий. В общем случае система может послать сообщение Heartbeat на доли секунд раньше, чем это сделает клиент, и такое поведение не должно рассматриваться как ошибочное. Таким образом, если мы создадим тест, точно следуя данному описанию технологического процесса, он будет находить ошибку, хотя поведение системы корректно. Очевидно, что представление не является полным, так как не включает, например, случай ошибки в клиентском сообщении Logon - в этом случае система не должна отвечать сообщением Logon.

В случае более сложных сценариев, текстовое описание технологического процесса может быть еще более неточным, неполным и к тому же громоздким, что лишь усложняет задачу написания тестов, так как возникает риск того, что отдельные части спецификации будут противоречить друг другу.

В дополнение к этим общим требованиям, необходимым и для ручного создания сценариев тестирования, автоматизированное создание тест-кейсов также подразумевает, что описание технологического процесса должно иметь формат, который может интерпретировать программа.

Насколько сложные сценарии могут быть указаны в технологическом процессе для тестирования протоколов? В предыдущей главе было сказано про связь системы и используемых ею протоколов. Рассмотрение каждой отдельной комбинации полей сообщений в технологическом процессе неизбежно требует анализа логики системы. Чем сложнее система, тем сложнее протокол. Наш опыт показывает, что для тестирования финансовых протоколов в большинстве случаев имеет смысл использовать самый общий уровень описания событий - ожидание только сообщений. В противном случае процесс написания технологического процесса становится слишком долгим. Таким образом, описание технологического процесса должно отвечать требованию лаконичности.

Итак, можно выделить следующие требования к описанию технологического процесса:

- Точность;
- Полнота;
- Лаконичность;
- Наглядность;
- Гибкость;
- Возможность чтения программой.

Для описания технологического процесса хорошо подходит концепция конечных автоматов (англ. Finite State Machine или FSM). Это вызвано тем, что конечные автоматы хорошо описывают реактивные, то есть реагирующие на события, системы ПО[5]. Программные комплексы, существующие для обеспечения нужд финансовых институтов, таких как биржи, центральные депозитарии, клиринговые системы или во всяком случае их части - торговые шлюзы, являются реактивными по своей сути. Так, FIX шлюз биржи пошлет сообщения клиенту только в ответ на его предыдущие сообщения. Существует множество способов описания FSM[6]: таблица "Состояние/Событие", представимая в виде csv-file, машина Мили (англ. Mealy machine), машина Мура (англ. Moore machine), UML, SDL, SCXML и прочие.

В работе[7] рассматривается создание сценариев тестирования протокола IRC на основе модели расширенного автомата, при этом используется описание в виде машины Мили и таблицы “Состояние/Событие”. Таблица “Состояние/Событие” достаточно наглядна при условии, что она небольшая. Такое представление может быть интерпретировано программой и позволяет легко отслеживать полноту описания технологического процесса и созданных тестов. Однако стоит заметить, что IRC протокол достаточно прост - как видно из статьи, его можно описать конечным автоматом с 4 состояниями и 7 событиями. При увеличении возможных состояний и/или событий, такой способ описания становится абсолютно ненаглядным. Кроме того, если система имеет конфигурируемые параметры, то при изменение параметра таблица должна быть значительно переделана. В случае использования недискретных переменных, например времени, требуется выбор какой-либо модели для тестирования уже на стадии описания.

Финансовые протоколы, как правило, гораздо сложнее, чем упомянутый IRC. Так, при попытке описания существующего технологического процесса реализации FIX протокола с помощью классической таблицы “Состояние/Событие” было обнаружено, что количество состояний увеличивается очень быстро.

Например, согласно спецификации[3] клиент должен быть отключен от системы, если пропустил 6 сообщений Heartbeat. Таким образом, чтобы правильно описать последствия события "время неактивности стало равным Heartbeat interval", нужно ввести 6 состояний "Пропущено 0 сообщений Heartbeat", "Пропущено 1 сообщение Heartbeat" и так далее. В то же самое время, сообщение посланное клиентом, может иметь порядковый номер меньше ожидаемого, равное ожидаемому или больше ожидаемого системой, итого 3 состояния для каждого сообщения. Общее количество сообщений, определенных в протоколе FIX, на данный момент равно 158[1]. Скорее всего, в конкретной системе не все они используются, но число используемых не меньше 10[3]. Параметры “ожидаемый порядковый номер” и “количество пропущенных сообщений Heartbeat” комплементарны по отношению друг к другу, то есть возможны любые комбинации значений этих параметров (стоит отметить, что некоторые события могут влиять на оба параметра). В результате общее количество возможных состояний равно произведению различных комплементарных параметров.

Рассматривая только эти параметры, мы получаем больше 6*3*10 = 180 различных состояний. Кроме того, в реальной системе максимальное количество пропущенных сообщений Heartbeat является конфигурируемым параметром. Поэтому, как было отмечено выше, описание в виде таблицы “Состояние/Событие” потребует значительной переработки в случае изменения параметра.

Многих недостатков, которые присущи представлению в виде таблицы “Состояние/Событие”, лишено представление FSM в унифицированном языке моделирования (англ. Unified Modeling Language - UML) [8, 9]. В нем, кроме прочего, предлагается использовать не состояния, а так называемые "расширенные состояния", которые включают в себя качественный аспект (само состояние) и количественный аспект (переменные "расширенного состояния"). Так, в нашем примере мы можем ввести переменную "количество пропущенных сообщений Heartbeat". Кроме того, можно ввести события, которые будут выполняться только при наступлении определенного условия (называемые “Guard condition” или просто “Guard”). Такое представление гораздо более наглядно и является более гибким.

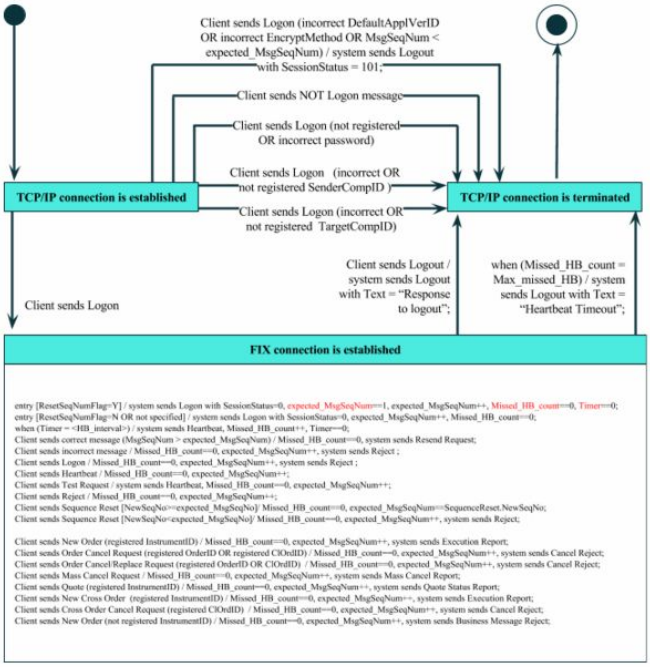

Рис. 3. Представление части технологического процесса реализации FIX протокола на UML-подобном языке

Пример описания технологического процесса реализации FIX протокола с помощью конечных автоматов на UML-подобном языке представлен на Рис. 3. Представление конечных автоматов в таком виде при всей наглядности обладает и недостатками. Полноту описания технологического процесса, представленного в виде UML, проверить сложнее, чем при описании таблицей “Состояние/Событие”. Но для нашей задачи главным недостатком представляется тот факт, что в UML-TP предлагается архитектура, нацеленная больше на ручную разработку тестов[15].

Существует XML спецификация для представления конечных автоматов, она называется SCXML[10]. Однако представление в таком виде, в свою очередь, не так наглядно как в языке UML.

Таким образом, каждый из способов описания конечного автомата имеет свои достоинства и недостатки. В разных ситуациях и для различных задач может понадобиться то или иное описание. Возможность преобразования одного вида описания в другой поможет объединить лучшие стороны каждого представления.

V. Программная реализация генерации сценариев тестирования

A. *Общие положения*

В тестировании на основании состояний применяются различные критерии покрытия. Слабейший критерий требует, чтобы тесты прошли каждое состояние и проверили каждый переход[14]. В нашей работе мы рассмотрим именно этот критерий. Попробуем описать возможный алгоритм работы приложения, которое, основываясь на технологическом процессе и словаре, создаст набор тестов, проверяющий протокол и каждый переход, указанный в технологическом процессе. Переходы мы разделим на 3 типа:

- Относящиеся к тестированию соединения. Такие переходы возникают при посылке правильных с точки зрения словаря сессионных сообщений, а также в сопутствующих сценариях;
- Относящиеся к позитивному тестированию протоколов. Эти переходы возникают при посылке правильных с точки зрения словаря прикладных сообщений;
- Относящиеся к негативному тестированию протоколов. Данные переходы возникают при посылке несоответствующих словарю сообщений, как административных, так и прикладных.

Желательно, чтобы тест-кейсы были независимы, то есть могли выполняться в любом порядке. Для реализации этого требуется, чтобы в начале тест-кейса система была в начальном состоянии, а в конце тест кейса в конечном. Таким образом, любой тест-кейс должен содержать шаги для перевода системы в конечное состояние, даже если проверяемый переход не приводит к этому.

Отметим, что начальное и конечное состояния зависят от конкретного технологического процесса. Они должны выбираться в зависимости от рассматриваемой области. Для тестирования соединения протоколов прикладного уровня, коими являются все финансовые протоколы, весьма логично рассматривать в качестве начального состояния “TCP/IP connection is established”, то есть TCP/IP соединение установлено, а в качестве конечного - “TCP/IP connection is closed” (TCP/IP соединение закрыто). Если бы мы рассматривали тестирование реализации протокола

TCP, начальное состояние было бы другим - например, “соединение установлено на сетевом уровне”. А для тестирования основной функциональности системы мы, наоборот, использовали бы начальное состояние в виде “соединение установлено на уровне прикладного (финансового) протокола” или еще более “высокие” состояния.

В данной статье описывается подход, где приложение последовательно рассматривает каждый переход, представленный в технологическом процессе, и генерирует тест-кейс или тест-кейсы для него. При этом, в результате использования переходов для вспомогательных целей, а именно перехода в начальное для данного события состояние, и перехода в конечное состояние, некоторые переходы могут проверяться многократно и даже приводить к абсолютному дублированию тестов. Однако, это позволит иметь простые тесты, которые могут служить индикаторами для решения о выполнении более сложных тестов. Часто не имеет смысла выполнять сложный тест, если он уже на первом шаге обнаруживает ошибку. Кроме того, в некоторых случаях это даст уверенность, что результат теста не зависит от пути. К тому же представляется, что такую логику работы программы легче реализовать. Рассмотрение оптимизации данного подхода является предметом будущих исследований.

Нашей конечной целью является определение общих подходов, которые будут применимы к автоматизированному созданию тест-кейсов для любого протокола. Для этого рассмотрим возможную работу приложения на конкретном примере компонента протокола распространения данных о котировках ITCH, отвечающего за восстановление утраченных сообщений[11]. На основе текстовой спецификации нами был сформирован технологический процесс, описывающий большую часть интересующей нас функциональности, а именно необходимой для тестирования соединения и протокола. Описание было выполнено в трех представлениях: UML-подобный язык, таблица “Состояние/Событие” и SCXML.

В соответствии со сказанным выше, начальным состоянием в технологическом процессе является состояние “TCP/IP connection is established” (TCP/IP соединение установлено), а конечным - “TCP/IP connection is closed” (TCP/IP соединение закрыто). Кроме того, в UML-подобном представлении, все события и действия, которые являются событиями, переводящими состояние само в себя, окрашены для удобства в 3 цвета, в зависимости от типа перехода. В этом случае мы сможем сгруппировать тесты, созданные для каждого типа, что в дальнейшем облегчит работу с ними. Мы разберем переходы последовательно и начнем с тех, которые относятся к тестированию соединения, затем рассмотрим те, что относятся к позитивному тестированию и, наконец, разберем переходы, относящиеся к негативному тестированию протокола.

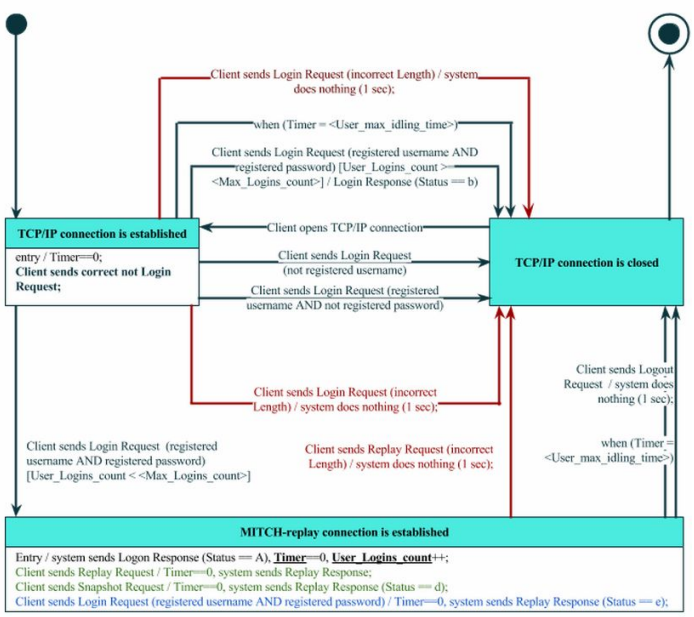

Рис. 4. Представление части технологического процесса реализации ITCH протокола на UML-подобном языке

B. Тестирование соединения

Приложение может выбирать переходы в любом порядке. Начнем, например, с “Client sends Login Request (not registered username)”. Рассмотрим детально описание этого события.

“Client sends” - тестировщик или инструмент для тестирования имитирует действия клиента. Поэтому это выражение означает, что что-то необходимо послать в систему, в отличие от “system sends/does”, где мы будем ждать определенного сообщения или действия от системы.

“Login Request” - название сообщения согласно словарю. Стоит заметить, что, если не указано обратное, сообщение считается корректным с точки зрения протокола.

“(not registered username)” - согласно принятой семантике языка UML в круглых скобках указываются атрибуты перехода.

“username” - это имя поля в сообщении Login Request. В нем, как следует из названия, указывается имя пользователя в системе. Для этого поля, также как для некоторых других полей, кроме деления на категории correct/incorrect, которые определяют правильность поля с точки зрения словаря, следует различать категории registered/not registered. Эти категории определены для полей, правильные значения которых ограничиваются не только протоколом, но и списком возможных значений соответствующего параметра в самой системе.

Поле является “Registered” (зарегистрированным, определенным), если его значение совпадает с одним из возможных значений соответствующего параметра в системе. Соответственно, поле является “Not registered” (противоположность Registered), если его значение не совпадает ни с одним из возможных значений соответствующего параметра в системе. В общем случае,

некоторые из значений параметра, зарегистрированного в системе, могут быть некорректными с точки зрения протокола. Однако, такая конфигурация должна рассматриваться как ошибочная, и потому на практике не должна встречаться. Поэтому в дальнейшем мы будем считать категории registered/not registered относящимися к категории correct. Соответственно, значение, некорректное с точки зрения протокола, автоматически будет считаться неопределенным в системе.

Итак, чтобы реализовать такое событие, в тест-кейсе должно быть послано сообщение Login Request, в котором значения во всех полях соответствуют словарю. Однако, значение поля username является незарегистрированным. Как следует из словаря, в теле сообщения Login Request 4 поля: Length, MsgType, username, password.

Поля Length и MsgType однозначно определяют сообщение, значения в них фиксированы, то есть каждое их них имеет только одно корректное значение.

Поле password имеет тип данных String, и о нем нет деталей в технологическом процессе, поэтому мы можем заполнить его любой правильной строкой. Отсюда следует, что наш инструмент для тестирования должен уметь генерировать соответствующее или несоответствующее словарю значение в зависимости от атрибутов поля, определенных в словаре. Назовем такую функцию #{RandomString(10)}, в скобках - длина строки.

Поле username должно иметь not registered значение. Выше мы обсудили значение понятия "not registered". Однако теперь необходимо подумать, как его задавать. Здесь возможны разные подходы, в том числе автоматическая генерация набора байт, ограниченного типом данных и другими атрибутами поля, заданными словарем. Однако существует ненулевая вероятность, что созданная строка будет являться "registered" полем. Более надежным и простым подходом видится задание "not registered" поля в виде конфигурируемого параметра - переменной. В зависимости от подхода, она может быть локальной (определенной только внутри тест-кейса) или глобальной. Назовем ее not_registered_username, а вызывать ее будем функцией %{not_registered_username}. Недостатком такого подхода является тот факт, что автоматически будет генерироваться только один тест-кейс.

Итак, входные данные готовы. Согласно технологическому процессу на выходе мы не ждем никаких сообщений от сервера, только переход в конечное состояние. Поэтому сгенерированный тест-кейс будет иметь вид наподобие следующего:

- Послать сообщение Login Request с атрибутами Length=18, MsgType=0x01, Password=#{RandomString(10)}, username = %{not_registered_username};
- Подождать некоторое время;
- Проверить, что TCP/IP соединение закрыто;
- Проверить, что количество пришедших сообщений от системы равно нулю за все время теста.

В технологическом процессе не задано время, необходимое для осуществления перехода, а значит теоретически он должен осуществляться мгновенно. В реальном случае время перехода равно совокупной задержке на упаковку, доставку, обработку сообщения шлюзом системы, поэтому мы добавили второй шаг. Это связано с тем, что нашей целью не является проверка времени отклика, и мы не хотим ложных ошибок теста из-за нее. В то же время, слишком большое ожидание увеличит время выполнения тест-кейса. Реальная задержка является функцией многих параметров, поэтому значение времени ожидания должно быть конфигурируемо. Назовем его timeout, а вызывать его будем функцией %{timeout}

Выходное состояние при данном событии совпадает с конечным, поэтому нет необходимости использовать другие переходы.

Переходим к следующему событию. “Client sends Login Request (registered username AND not registered password)”

В атрибутах этого поля есть ключевое слово AND, которое является обычным логическим “И”. Реализация этого события похожа на предыдущую, за исключением того, что поле password должно быть not registered, а username, напротив - registered. Для registered полей будем использовать тот же подход с указанием значения через вызов переменной. Для поля username переменную назовем registered_username. Итого, сгенерированный тест-кейс будет иметь вид наподобие следующего:

- Послать Login Request с атрибутами Length=18, MsgType=0x01, Password = %{not_registered_password}, username = %{registered_username};
- Подождать %{timeout};
- Проверить, что TCP/IP соединение закрыто;
- Проверить, что количество пришедших сообщений от системы равно нулю за все время теста.

Рассмотренные выше события должны происходить при определенных действиях со стороны клиента. Такие события в языке UML называются “событиями вызова” (англ. call event).

Рассмотрим следующее событие “when (Timer=<User_max_idling_time>)”. Это событие означает, что когда значение глобальной системной переменной Timer становится равным заданному системному параметру <User_max_idling_time>, то есть “максимальному времени бездействия”, система должна закрывать TCP/IP соединение с клиентом. Такое событие отличается от рассмотренных выше тем, что оно может произойти без каких-либо действий со стороны клиента. В языке UML такой вид событий называется “событием изменения” (англ. change event). И семантика для его обозначения отличается от событий вызова. Оно начинается с ключевого слова “when", и в скобках указывается булево выражение, которое вызывает событие, когда его значение становится TRUE. В отличие от “Guard condition”, работа которых будет рассмотрена ниже, вычисление булевого выражения производится всегда, а не только когда данное событие вызывается.

Тесты, созданные для проверки этого события, должны проверять оба варианта, когда булево выражение равно

TRUE и когда оно равно FALSE. В обоих случаях система не должна генерировать никаких сообщений. Можно реализовать обе проверки в одном тест-кейсе, он будет содержать пять шагов:

- Подождать время равное <User_max_idling_time>/2;
- Проверить, что TCP/IP соединение открыто;
- Подождать время равное <user_max_idling_time>/2+%{timeout};
- Проверить, что TCP/IP соединение закрыто;
- Проверить, что количество пришедших сообщений от системы за все время теста равно нулю.

Переходим к событию “Client sends Login Request (registered username AND registered password) [User_Logins_count < <Max_Logins_count>]”. Оно является событием вызова, но, кроме указания действия и его атрибутов, содержит квадратные скобки, в которых заключено булево выражение. Это условие называется “Защитник” (англ. Guard). Когда действие происходит, система проверяет, что Защитник имеет значение TRUE и только в этом случае происходит переход и ответная реакция от системы. В нашем случае в условии проверяется, что количество успешно выполненных входов в систему меньше максимально допустимого. Для этого используются системная переменная “User_Logins_count” и системный параметр <Max_Logins_count>. В данном построении мы можем проверить переход, только если значение Защитника равно TRUE, если же его значение равно FALSE, этот переход не будет осуществлен. Должно существовать другое событие, которое будет реализовываться, если Защитник имеет значение FALSE (очевидно, оно также будет содержать Защитника, но с противоположным условием). В нашем случае этим другим событием является “Client sends Login Request [User_Logins_count >= <Max_Logins_count>] / Login Response (Status == b)”, мы рассмотрим его позже. Вернемся к событию “Client sends Login Request (registered username AND registered password) [User_Logins_count < <Max_Logins_count>]”. Предполагая, что условие выполнено, а для этого на этапе предварительной подготовки мы должны удостовериться что User_Logins_count == 0, нам остается выполнить простое действие “Client sends Login Request (registered username AND registered password)”.

Оно переводит нас в состояние “MITCH-replay connection is established”. Это состояние имеет событие, которое должно выполняться как только система переходит в это состояние: “entry / system sends Logon Response (Status == A), Timer==0, User_Logins_count++;”, называемого “действием при входе” (англ. entry action). Указателем на этот факт является ключевое слово “entry”. В соответствии с семантикой языка UML после знака слэш “/” указаны действия, которые должна сделать система. Вообще говоря, они все должны быть проверены. “User_Logins_count++” означает, что система должна увеличить значение переменной “User_Logins_count” на единицу. Проверка этого очевидно связана с обсуждавшимися ранее Защитниками. “Timer==0” указывает, что значение переменной Timer должно быть установлено в ноль. Вообще, для каждой системной переменной, упомянутой в технологическом процессе, должен существовать как минимум один переход, использующий ее значение. Кажется разумным все изменения системной переменной учитывать в тестах, рассматривающих переход, где участвует эта переменная. Таким образом, осталось проверить реакцию системы “system sends Logon Response (Status == A)”. Как было сказано выше, ключевые слова “system sends” означают, что мы должны ждать сообщения от системы. Атрибут “Status == A” сообщает, что следует ожидать значение “A” в поле Status. Ожидаемые значение других полей должны быть ограничены приложением при помощи словаря. Согласно словарю, кроме поля Status сообщение Login Response содержит также поля Length и MsgType. Также как для сообщения Login Request, они могут иметь единственные корректные с точки зрения словаря значения, именно их мы должны ждать в сообщении от системы. Кроме того, состояние “MITCH-replay connection is established” не является конечным, поэтому должны быть добавлены шаги для перевода системы в конечное состояние. Инструмент для тестирования должен выбрать любой из имеющихся путей перехода в конечное состояние. Не уменьшая общности, рассмотрим, что был выбран переход “Client sends Logout Request / system does nothing (1 sec);”. Согласно словарю, Logout Request состоит всего из двух полей, Length и MsgType, с которыми наше приложение уже умеет работать. Способ проверки реакции системы легко представить из сказанного выше. Итоговый тест-кейс может выглядеть следующим образом:

- Послать Login Request с атрибутами Length=18, MsgType=0x01, Password = %{registered_password}, username = %{registered_username}.
- Ожидать в течение времени %{timeout} сообщение Login Response с атрибутами Length=3, MsgType=0x02, Status = A.
- Проверить, что TCP/IP соединение открыто
- Послать Logout Request с атрибутами Length=2, MsgType=0x05
- Подождать время равное ½ секунды
- Проверить, что TCP/IP соединение открыто
- Подождать время равное ½ секунды + %{timeout}
- Проверить, что TCP/IP соединение закрыто
- Проверить, что количество пришедших сообщений от системы за все время теста равно одному.

Рассмотрим далее событие “Client sends Logout Request / system does nothing (1 sec);”. Оно переводит систему из состояния “MITCH-replay connection is established”, не являющегося начальным, в конечное состояние. Поэтому тест-кейс должен содержать шаги для перевода системы из начального состояния в необходимое. Приложение должно выбрать какой-либо путь, в нашем случае для этого существует единственный переход - “Client sends Login

Request (registered username AND registered password) [User_Logins_count < <Max_Logins_count>]". Таким образом, сгенерированный тест будет абсолютно идентичен предыдущему, нет необходимости описывать его снова. О том, что в результате нашего подхода могут появляться абсолютно идентичные тест-кейсы, мы уже упоминали во время обсуждения общих положений.

Событие "Client sends Login Request (registered username AND registered password) / Timer==0, system sends Replay Response (Status == e);" интересно тем, что оно является примером "внутреннего перехода" (англ. internal transition), которое похоже на "переход само-в-себя" (англ. self-transition), с той разницей, что не должны выполняться "действие при входе" и "действие при выходе". Кроме того, в нем генерируется сообщение Replay Response, которое мы еще не рассматривали. Согласно словарю, это сообщение имеет шесть полей: Length, MsgType, MarketDataGroup, FirstMessage, Count, Status. Ожидаемое значение поля Status указано в технологическом процессе, работа с полями Length и MsgType разобрана выше. Значение поля MarketDataGroup не ограничено ни словарем, ни технологическим процессом. Интерес представляют поля FirstMessage и Count, которые, согласно спецификации, должны "иметь значение 0, если Status не равен A". Такое ограничение должно быть задано на уровне словаря, его можно реализовать в языке XSD. Наш инструмент для тестирования должен уметь обрабатывать такие ограничения. В остальном составление тест-кейса для данного события не должно вызвать осложнений, так как остальные его части были разобраны при рассмотрении других событий. Тест-кейс будет состоять из трех частей:

- Переход из начального состояния в состояние "MITCH-replay connection is established";
- Выполнение самого события и ожидание реакции от системы;
- Переход из состояния "MITCH-replay connection is established" в конечное состояние.

Событие "Client sends correct not Login Request;". Это событие является обобщением нескольких событий, объединенных в нашем представлении для удобства. Мы можем это сделать, потому что состояние, в котором происходит событие, и реакция системы одинаковы для всех них. В описании технологического процесса это задается конструкцией "not Login Request", состоящей из булевого оператора NOT и имени сообщения Login Request. Отметим также ключевое слово "correct", означающее что сообщение должно быть корректным с точки зрения словаря. Заметим, что сообщение должно считаться корректным только в том случае, если оно определено для посылки клиентом. Поэтому представляется необходимым, чтобы каждое сообщение в словаре имело атрибут, который бы указывал, определено ли сообщение для посылки клиентом или сервером. Для упрощения можно договориться о том, что если атрибут не указан, считать, что сообщение определено для посылки обеими сторонами. Итак, инструмент тестирования ищет в словаре все сообщения, которые определены для посылки клиентом, но не являются сообщением Login Request, и генерирует их в соответствии только со словарем, так как атрибуты не указаны в технологическом процессе. В нашем случае такими сообщениями являются Logout Request, Replay Request, Snapshot Request. Это событие является внутренним, без какой-либо реакции со стороны системы. Как обычно, каждый тест-кейс должен заканчиваться переходом в конечное состояние. Примеры сгенерированных тест-кейсов:

- Послать сообщение Snapshot Request с атрибутами Length=30, MsgType=0x81, SequenceNumber = #{RandomUInt32}, Segment = #{RandomString(6)}, InstrumentID = #{RandomUInt32}, SubBook = 0, SnapshotType = 1, RecoverFromTime = #{RandomTime}, RequestID = #{RandomUInt32};
- Подождать %{timeout};
- Проверить, что TCP/IP соединение открыто;
- Послать Login Request с атрибутами Length=18, MsgType=0x01, Password = %{registered_password}, username = %{registered_username};
- Ожидать в течение времени %{timeout} сообщение Login Response с атрибутами Length=3, MsgType=0x02, Status = A;
- Проверить, что TCP/IP соединение открыто;
- Послать Logout Request с аттрибутами Length=2, MsgType=0x05;
- Подождать время равное ½ секунды;
- Проверить, что TCP/IP соединение открыто;
- Подождать время равное ½ секунды + %{timeout};
- Проверить, что TCP/IP соединение закрыто;
- Проверить, что количество пришедших сообщений от системы за все время теста равно одному.

В тестировании соединения осталось проверить реализацию событий, включающих в себя работу с системными переменными.

Переход "when (Timer = <User_max_idling_time>);" из состояния "MITCH-replay connection is established" в состояние "TCP/IP connection is closed". В первом тест-кейсе будет проверка самого перехода, она будет состоять из двух частей:

- шаги для перевода системы в состояние "MITCH-replay connection is established";
- шаги для проверки перехода "when (Timer = <User_max_idling_time>);", мы их разбирали выше.

Последующие тест-кейсы должны включать в себя проверки изменения системной переменной в случае событий, описанных в технологическом процессе. Они будут состоять из следующих частей:

- Шаги для перевода системы в состояние "MITCH-replay connection is established";
- Подождать время равное <User_max_idling_time>/2;

- Шаги, осуществляющие событие, изменяющее значение переменной Timer;
- Шаги для проверки перехода “when (Timer = <User_max_idling_time>);”.

В нашем примере все внутренние переходы (их три) внутри состояния “MITCH-replay connection is established” изменяют значение переменной Timer. Таким образом, в дополнении к основному тест-кейсу, мы будем иметь три дополнительных.

Осталось 2 последних перехода в тестировании соединения: “Client sends Login Request (registered username AND registered password) [User_Logins_count >= <Max_Logins_count>] / Login Response (Status == b)” и “Client opens TCP/IP connection”. Они оба связаны с системной переменной User_Logins_count. Второй переход является вспомогательным, он определен только для того, чтобы осуществить первый (нет проблемы в том, что по общей логике работы приложения будет создан отдельный тест-кейс для проверки второго перехода). Общая структура тест-кейса, проверяющего переход с Защитником, будет следующая:

- Шаги для перевода системы в состояние “MITCH-replay connection is established”;
- Шаги для перевода системы в состояние “TCP/IP connection is closed”;
- Переход в начальное состояние “TCP/IP connection is established”;
- Повторить шаги 1)-3) <Max_Logins_count> раз;
- Послать Login Request с registered username и registered password;
- Ожидать сообщения Login Response с Status = b.

Обобщая примеры событий, работающих с системными переменными, следует сказать, что инструмент для тестирования должен уметь проверять работу переменных. Для этого он должен выделять события, использующие переменные (события с Защитником и события изменения), а также события, которые меняют значения переменных, и составлять на их основе комплексные тест-кейсы.

C. Позитивное тестирование протоколов

Проверка событий, относящихся к позитивному тестированию протоколов, похожа на проверку событий для тестирования соединения. Разница в том, что обычно прикладных сообщений, являющихся предметом рассмотрения позитивного тестирования протоколов, больше, и количество полей в них также больше, нежели в административных сообщениях, которые рассматриваются в тестирования соединения. Однако в нашем случае определено всего два прикладных сообщения: Replay Request и Snapshot Request. Поэтому и количество событий равно двум. Мы не будем подробно описывать детально способ генерации тест-кейсов, так как большинство приемов, использующихся в тестировании соединения должны быть использованы и здесь. Остановимся только на особенностях.

“Client sends Replay Request / Timer==0, system sends Replay Response;”. Мы уже посылали сообщение Replay Request, когда проверяли событие “Client sends not Login Request message”, однако в том случае мы удовлетворились одним тест-кейсом для данного сообщения. В данном же случае, напомним, целью является проверка работы системы в случае получения различных комбинаций корректных с точки зрения словаря сообщений. Как было отмечено в главе 3, представляется разумным использовать широко распространенные подходы к тестированию, такие как классы эквивалентности, для определения достаточного набора различных сообщений. Для каждого сообщения должен быть сгенерирован отдельный тест-кейс.

Некоторые примеры различных комбинаций значений:

- MarketDataGroup = <space>, FirstMessage=0, Count=0;
- MarketDataGroup = <NULL>, FirstMessage=0, Count=65535;
- MarketDataGroup = A, FirstMessage = 20000, Count = 5678;
- MarketDataGroup = b, FirstMessage = 1000, Count = 20000;

D. Негативное тестирование протоколов

Как и для позитивного тестирования протокола, рассмотрим лишь особенности.

“Client sends Login Request (incorrect Length) / system does nothing (1 sec);” Ключевое слово “incorrect” в указании события означает, что поле или сообщение некорректно с точки зрения протокола. Если в workflow не уточнено как именно, то, подразумевается, что любой вид ошибки может быть использован. В нашем примере слово “incorrect” указано в атрибуте и означает, что значение в поле Length должно быть неправильным. В остальном сообщение должно соответствовать протоколу. Инструмент тестирования должен уметь генерировать сообщение для каждого возможного неправильного сообщения, подходы могут отличаться в зависимости от конкретного протокола. Для нашего примера может быть использован следующий алгоритм:

- Генерируется корректное сообщение Login Request;
- Изменяется значение в поле Length;
- Если новое значение больше правильного, произвольные байты добавляются после последнего байта сообщения, сгенерированного на первом шаге.

VI. Правила создания сообщений на основе словаря

Инструмент для тестирования должен уметь генерировать корректные и некорректные сообщения на основе словаря. Для этого необходимо определить метод создания позитивных и негативных тестов для каждого атрибута каждого поля, заданного в словаре. Примеры

атрибутов и соответствующие им позитивные и негативные тесты приведены в Таблице II.

ТАБЛИЦА II. ОПИСАНИЕ НАБОРА ТЕСТОВ ДЛЯ РАЗНЫХ АТРИБУТОВ

Аттрибут	Позитивные тесты	Негативные тесты
MinOccurs	Количество появлений поля в сообщении больше либо равно параметру MinOccurs	Количество появлений поля в сообщении меньше параметра MinOccurs
MaxOccurs	Количество появлений поля в сообщении меньше либо равно параметру MaxOccurs	Количество появлений поля в сообщении больше параметра MaxOccurs
Length	Длина поля меньше либо равна параметру Length	Длина поля больше параметра Length
Min Value	Значение поля больше либо равно MinValue	Значение поля меньше MinValue
Max Value	Значение поля меньше либо равно MaxValue	Значение поля больше MaxValue

Большое влияние на количество тест-кейсов оказывает такой обязательный атрибут, как тип данных. Рассмотрим некоторые типы данных и примеры связанных с ними тестов (заметим, что для бинарных протоколов негативные тесты неприменимы):

A. Строки

1) Позитивные:

- Строка заполнена на всю допустимую длину печатными символами, нет пробелов;
- Строка заполнена печатными символами, за которыми следуют пробелы;
- Строка заполнена печатными символами и пробелами случайным образом.

2) Негативные:

- Строка содержит непечатные символы.

B. Целые числа

1) Позитивные:

- Значение в середине диапазона;
- Значение равно максимальному/минимальному для этого типа данных;
- Значение с нулями вначале, например “0100” (специфичен для текстового протокола).

2) Негативные:

- Значение содержащие символы кроме “-” и “0-9”;
- Значение с символом “-”, находящемся не в начале.

C. Числа с плавающей точкой

1) Позитивные:

- Значение в середине диапазона;
- Значение равно максимальному/минимальному для этого типа данных;
- Значение с максимально возможным количеством знаков после запятой;
- Значение с нулями в начале, например “05” (специфичен для текстового протокола);
- Значение с нулями после десятичной части, например “1.12000” (специфичен для текстового протокола);
- Значение с точкой в начале, например “.01” (специфичен для текстового протокола);
- Значение с точкой в конце, например “345.” (специфичен для текстового протокола);

2) Негативные:

- Значения имеющие в себе символы кроме “-”,“.” и “0-9”;
- Значения с символом “-”, находящимся не в начале.

Резюмируя, можно выделить следующие требования к инструменту для тестирования:

- Умение интерпретировать семантику языка представления технологического процесса, в частности интерпретирование атрибутов, таких как registered, correct, send и прочих;
- Наличие алгоритма выбора пути перехода в конечное состояние;
- Умение генерировать соответствующее или несоответствующее словарю поле и/или сообщение;
- Умение проверять работу системных переменных, определенных в технологическом процессе.

ЗАКЛЮЧЕНИЕ

Инструменты и методы автоматизированного создания тестов как на основе словарей и так и на основе применения конечных автоматов достаточно подробно описаны в научной литературе[7,12,15,16]. В данной статье мы попытались проанализировать совмещение обоих подходов и его применимость к динамической верификации финансовых протоколов. В рамках исследования были рассмотрены компоненты протокола распространения данных о котировках ITCH, отвечающие за восстановление утраченных сообщений. Разработано представление конечного автомата, задающего технологический процесс на UML-подобном языке и на языке SCXML. Уточнены требования к описанию словарей и технологическому процессу, необходимые для автоматизированного создания тестов.

Дальнейшими направлениями работы будут:

- Более глубокий анализ существующих способов описания технологического процесса с помощью конечных автоматов;
- Анализ других финансовых протоколов для уточнения требований к инструменту для тестирования;
- Построение прототипа инструмента для тестирования;
- Проверка подходов, заложенных в требованиях к инструменту и решений для их реализации;
- Нахождение фундаментальных ограничений на применение данного подхода.

Источники

[1] Technical Resources / Specifications [электронный ресурс] // FIX Trading Community. URL: http://www.fixtradingcommunity.org/pg/structure/tech-specs (дата обращения: 15.09.2015)

[2] About ISO 20022 [электронный ресурс] // ISO 20022 Universal financial industry message scheme. URL: http://www.iso20022.org/about_iso20022.page (дата обращения: 21.09.2015)

[3] Technical Specifications - Live Version MIT202 - Trading Gateway (FIX 5.0) Specification - Issue 7.2 [электронный ресурс] // Borsa Italiana. URL: http://www.borsaitaliana.it/borsaitaliana/gestione-mercati/migrazionemillenniumit-mit/mit202-tradinggatewayspecification-issue71_pdf.htm (дата обращения: 10.09.2015)

[4] FIIDL - FIX Interactive Interface Definition Language [электронный ресурс] // J. Greenan. URL: http://blog.alignment-systems.com/2014/11/fix-interactive-interface-definition.html (дата обращения: 13.09.2015)

[5] Harel D., Politi M. Modeling Reactive Systems with Statecharts. NY: McGraw-Hill, 1998

[6] Finite-state machine [электронный ресурс] // Wikipedia.org. URL: https://en.wikipedia.org/wiki/Finite-state_machine (дата обращения: 09.09.2015)

[7] М.В. Жигулин, А.В. Коломеец, Н.Г. Кушик, А.В. Шабалдин. Тестирование программной реализации протокола IRC на основе модели расширенного автомата. Известия Томского политехнического университета. 2011. Т. 318. No 5

[8] UML state machine [электронный ресурс] // Wikipedia.org. URL: https://en.wikipedia.org/wiki/UML_state_machine (дата обращения: 09.09.2015)

[9] UML specification [электронный ресурс] // omg.org. URL: http://www.omg.org/spec/UML/ (дата обращения: 11.09.2015)

[10] State Chart XML specification [электронный ресурс] // W3.org. URL: http://www.w3.org/TR/scxml/ (дата обращения: 25.09.2015)

[11] Technical Specifications - Live Version MIT303 – Market Data Feed (MITCH) Specification - Issue 7.1 [электронный ресурс] // Borsa Italiana. URL: http://www.borsaitaliana.it/borsaitaliana/gestione-mercati/migrazionemillenniumit-mit/mit303-marketdatafeedspecification-issue71_pdf.htm (дата обращения: 10.09.2015)

[12] A. Alexeenko, P. Protsenko, A. Matveeva, I. Itkin, D. Sharov. Compatibility Testing of Protocol Connections of Exchange and Broker Systems Clients, Tools & Methods of Program Analysis 2013

[13] Международный стандарт ISO/IEC 7498-1 [электронный ресурс] // International Organization for Standardization. URL: http://standards.iso.org/ittf/licence.html (дата обращения: 23.09.2015)

[14] Применение методик тестирования. Часть 4 [электронный ресурс] // Инрэко ЛАН. URL: http://inrecolan.ru/blog/viewpost/360 (дата обращения: 01.09.2015)

[15] А.В. Баранцев, И.Б. Бурдонов, А.В. Демаков, С.В. Зеленов, А.С. Косачев, В.В. Кулямин, В.А. Омельченко, Н.В. Пакулин, А.К. Петренко, А.В. Хорошилов. Подход UniTesK к разработке тестов: достижения и перспективы. [электронный ресурс] // Труды Института системного программирования РАН. URL: http://citforum.ru/SE/testing/unitesk/ (дата обращения: 03.09.2015)

[16] Никешин А. В., Пакулин Н. В., Шнитман В. З.. Разработка тестового набора для верификации реализаций протокола безопасности TLS. Труды Института системного программирования РАН том 23 / 201

[17] Дуглас Э. Камер. Сети TCP/IP, том 1. Принципы, протоколы и структура. М.: «Вильямс», 2003. — С. 880