Проверка соответствия SDL-диаграмм MSC-документации при имеющихся отличиях

В.В.Соколов
svv@tepkom.ru
Санкт-Петербургский государственный университет
198504, Университетский пр., 28
Санкт-Петербург, Россия

Аннотация

Современные телекоммуникационные системы и системы реального времени создаются на основе групп стандартов, включающих MSC и SDL. SDL- и MSC-диаграммы описывают взаимодействие объектов. Рассматривается задача проверки соответствия SDL-диаграмм MSC-документации. Формулируется проблема отличия диаграмм от документации. Дается обзор существующих подходов. Предлагается свой алгоритм проверки при априорном знании о наличии различий.

Введение

К телекоммуникационным системам и системам реального времени предъявляются повышенные требования по выдерживанию временных характеристик, гарантированному поведению в любых ситуациях и другие жесткие условия, вытекающие из природы задачи. Примером таких систем являются банковские системы, телефонные станции, спутники, системы обеспечения жизнедеятельности — во всех этих случаях слишком высока цена ошибки. Поэтому при разработке подобных систем крайне важны подробные спецификации, описывающие как можно большее количество деталей. Соответствующие спецификации преимущественно разрабатываются в стандартах MSC (Message

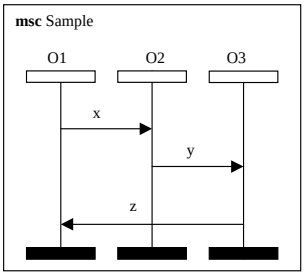

Рис. 1: MSC-диаграмма

Sequence Charts) [10, 8], SDL (Specification and Description Language) [9], UML (Unified Modeling Language) [14, 2]. Сейчас все три стандарта сливаются в один под названием UML 2.0. Данные стандарты являются определяющими в области телекоммуникационных систем для обеспечения взаимодействия средств, разработанных различными производителями. Поэтому стандарты на протоколы взаимодействия выпускаются в виде SDL- и MSC-диаграмм.

Хотя и SDL, и MSC имеют средства описания декомпозиции системы на части и описания деталей будущей реализации, их основной задачей является описание динамики поведения системы, и при упоминании этих стандартов мы будем подразумевать именно ее.

Традиционно MSC-диаграммы возникают на этапе проектирования системы. Пример MSC-диаграммы приведен на рисунке 1. На нем изображены три объекта и обмен сообщениями между ними. На MSC-диаграммах прорабатываются основные схемы взаимодействия. При этом данные схемы часто не обладают законченностью: некоторые второстепенные объекты, участвующие во взаимодействии, могут быть опущены; может опускаться цикличность обменов и вместо этого изображается лишь один вариант обмена; не указывается стыковка между собой различных диаграмм взаимодействия, даже если они описывают один объект. Возможны и другие различия. Основное требование, которое предъявляется при созданиии подобных схем — это отразить основные варианты взаимодействия объектов.

SDL-диаграммы, наоборот, описывают полное поведение объ-

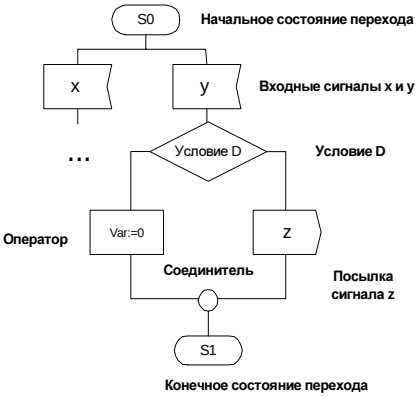

Рис. 2: Пример SDL-диаграммы

екта во всех ситуациях. Они являются комбинацией конечно-автоматной модели и блок-схем. Пример SDL-диаграммы приведен на рисунке 2. Данное описание является настолько подробным, что SDL-диаграммы, дополненные описанием на АЯВУ, относятся уже к этапу программирования. В современных системах они являются реализацией и имеют отображение в исполняемый код.

Фактически, эти два типа диаграмм описывают одно и то же — обмен сообщениями, только дают на него взглянуть с разных точек зрения. MSC-диаграммы показывают картину с точки зрения стороннего наблюдателя. При этом видно поведение объектов в совокупности, но не ясно, по каким принципам каждый из объектов принял то или иное решение. SDL-диаграммы, наоборот, изображают картину с точки зрения одного объекта, показывают на основе чего он выбрал тот или иной вариант поведения, но рассматривается поведение только одного объекта и нет информации о поведении других объектов в этот же момент времени.

Когда в жизненном цикле возникают два типа диаграм, которые описывают одно и то же, их несогласованность может быть причиной многих ошибок. Поэтому появляется проблема сравнения двух моделей. Данная задача возникает как при начальном

создании SDL- и MSC-диаграмм, так и в процессе разработки при изменениях моделей для того, чтобы знать, не появилось ли в них противоречий.

1 Существующие подходы

Мы рассмотрим задачу сравнения SDL-реализации и MSC-документации.

Точного решения здесь предложить нельзя, поскольку, во-первых, SDL- и MSC-модели являются моделями разных формальных уровней. При этом то, что на MSC записано в виде комментариев на естественном языке, на SDL записано в виде работы с переменными. Во-вторых возможность использования конструкций общего вида на SDL ведет к алгоритмической неразрешимости задачи.

При этом необходимость их сравнения важна и обе модели сильно похожи. И та и другая задают некоторую расширенную конечно-автоматную модель, алфавитом которой являются сигналы. По сути, надо сравнить порождаемые языки этих конечно-автоматных моделей и сформулировать результат проверки.

В существующих подходах для начала делается предположение, что доверие к MSC-диаграммам больше. Они являются эталоном, а SDL-диаграммы являются проверяемыми. Для проверок используются следующие подходы, которые условно можно разделить на три категории:

1. Из MSC-диаграммы объекта генерируются трассы — линейные последовательности сигналов. После этого данные последовательности “накладываются” на SDL-модель объекта, начиная с некоторой точки. При наложении сигналы должны быть одинаковыми и пропускать их нельзя. При наложении значения переменных, их анализ и операции с ними не учитываются. Поскольку на MSC можно описывать циклы, то потенциально набор трасс бесконечен. Мы ограничиваем себя в выборе конечного набора, например, по максимальной длине трассы, и проверяем лишь его. Если мы сумели наложить все эти трассы, то считается, что SDL-модель объекта соответствует его MSC-описанию. Если подобная проверка прошла успешно для всех объектов, то считается,

что MSC-описание системы соответствует SDL-описанию системы.

2. Модификация предыдущего подхода, когда разрешено пропускать сигналы на проверяемой модели [16].

3. Параллельное исполнение SDL- и MSC-моделей как двух белых ящиков. При исполнении принимаемые и порождаемые цепочки сигналов должны быть одинаковыми. Каждая из моделей может состоять из набора автоматов, исполняемых параллельно [7]. В идеальном варианте параллельно исполняется сразу вся SDL-система и вся MSC-система. Использование полного выполнения систем или введение каких-либо ограничений зависит от объема системы и количества доступных ресурсов. В процессе выполнения либо обнаруживаются расхождения в цепочках сигналов, либо, с учетом поставленных ограничений на перебор, считается, что две системы соответствуют друг другу.

Более детально о философии проверки соответствия MSC- и SDL-диаграмм можно прочитать в стандартах серии ETSI [5, 3, 4], а о реализации подходов 1 и 3 — в документации на средство Telelogic Tau [17].

Подход 2 не получил широкого распространения из-за недостоверности результатов, а подходы 1 и 3 достаточно экстремистичны в своих требованиях. Например, условие на последовательное совпадение сигналов слишком сильное. Оно нарушается простым добавлением нескольких сигналов для накопления статистики еще одним объектом. Так же при создании систем его можно нарушить еще десятком способов. За всем этим можно следить, параллельно редактируя не одну, а сразу обе модели, но в реальных проектах SDL и MSC так и остаются только похожими.

2 Постановка задачи

Пусть в процессе разработки системы мы получили MSC- и SDL-модели. Полного соответствия между ними нет. Причиной этого является трудность создания и поддержания одинакового

поведения сразу и на MSC, и на SDL. Про модели доподлинно известно, что они *описывают одну систему* с возможными "незначительными" изменениями, такими, как:

- опускание цикличности на MSC-диаграммах;
- реализации на SDL-модели нескольких MSC-ролей;
- введение дополнительных сигналов на SDL-модели;
- существование "дополнительных" объектов на SDL, опущенных на MSC-диаграммах;
- разбиение MSC-роли на несколько сервисов;
- отсутствие объединения различных MSC-сценариев одного объекта в единое целое;
- отсутствие на MSC-диаграммах символа завершения существования объекта.

Ставится задача проанализировать MSC-документацию и SDL-реализацию именно в таком виде с выявлением возможных ошибок.

Утверждается, что подобная ситуация появляется достаточно часто в жизненном цикле систем и при этом существующие алгоритмы проверки моделей на соответствие оказываются малоэффективными из-за того, что практически всегда SDL-модель "не совпадает" с MSC-моделью. Автор предлагает "зайти с другого конца" и проверять, что в SDL-модели нет противоречий с MSC-документацией. Таким образом, моделям разрешено быть различными, но не должно быть ситуаций, когда они друг другу противоречат. Разумеется, предлагаемая проверка может быть использована с уже существующими методами, описанными в предыдущем разделе.

При построении алгоритма используются следующие соображения:

1. мы считаем MSC-диаграммы априори достоверными, а SDL-диаграммы — проверяемыми;
2. мы отказались от анализа переменных на SDL и комментариев на MSC. В качестве информации с одной и с другой моделей мы используем конечно-автоматные модели, описывающие языки, алфавитом которых являются сигналы;

3. мы проверяем, что у SDL-реализации есть хотя бы одна возможность правильно обработать MSC-спецификацию. Это означает, что если у системы есть два варианта выполнения, один из которых ведет к ошибке, а другой — к положительному результату, то мы выбираем "положительный". Например, когда сигнал нужен и может придти вовремя, а может придти заранее и уничтожится из-за несохранения в состоянии, то мы выберем случай случай, когда он придет вовремя;

4. мы считаем ошибкой ситуацию, когда существует трасса с MSC-спецификации, которая не имеет отображения в SDL-реализацию. Как бы мы корректно ни исполняли программу, все равно она не соответствует спецификации;

5. возможно, что программа будет обрабатывать еще какие-либо трассы или из-за ошибок в программе будет выполнять лишь часть всех вариантов поведения. В данном подходе мы рассматриваем только принципиальную возможность выполнения с точки зрения ограничений, введенных в п. 2.

Таким образом, из SDL- и из MSC-моделей мы извлекаем конечно-автоматную структуру. В следующем разделе мы опишем, на основе чего принимается решение о противоречивости/непротиворечивости данных конечно-автоматных структур. После этого будет дано объяснение смысла полученного результата в терминах исходных моделей. Если конечно-автоматные структуры друг другу не противоречат, то же заключение мы делаем и для моделей, если противоречат, то считается, что модели также друг другу противоречат.

3 Формальное описание метода

Для начала дадим несколько математических определений, с помощью которых и сформулируем метод.

Определение 1. *Будем называть конечным автоматом набор* $M = (Q, V, T, E, S, F)$.

- Q — *конечное множество состояний,*

- V — *конечный алфавит,*
- T — *множество правил перехода,* $T \subseteq Q \times V \times Q$,
- E — *множество* ε*-переходов,* $E \subseteq Q \times Q$,
- S — *начальное состояние,* $S \in Q$,
- F — *множество конечных состояний,* $F \subseteq Q$.

Не умаляя общности, мы считаем, что множество пустых переходов доопределяется следующим образом:

- $\forall q \in Q : (q, q) \in E$,
- вместо E рассматривается E^* — для каждой вершины строится транзитивное замыкание ε-переходов.

В дальнейшем, вводя автомат M_i, мы будем подразумевать существование соответствующим образом проиндексированной шестерки $(Q_i, V_i, T_i, E_i, S_i, F_i)$.

Мы будем допускать вольность в использовании множеств T и E, считая, что существуют одноименные отображения $T : Q \times V \to 2^Q$ и $E : Q \to 2^Q$.

Определение 2. *Мы расширяем отношение/отображение* $T \subseteq Q \times V \times Q$ *до* $T^* \subseteq Q \times V^* \times Q$ *следующим образом:*

1. $\forall q \in Q : T^*(q, \varepsilon) = E(q)$,
2. $\forall a \in V, w \in V^*, q \in Q : T^*(q, aw) = E(T(T^*(q, w), a))$,

где V^* обозначает множество всех слов, состоящих из произвольных последовательностей над $(V \cup \varepsilon)$, с соответствующими правилами для пустого символа ε.

Определение 3. *Автомат называется* S*-корректным, если*

$$\forall q \in Q \ \exists w \in V^* : T^*(S, w) = q$$

Определение 4. *Автомат называется* V*-корректным, если*

$$\forall v \in V \ \exists (q_1, v, q_2) \in T$$

В дальнейшем мы будем рассматривать только V-корректные автоматы.

Определение 5. *Будем называть языком конечного автомата*

$$Lang(M) = (w \in V^* | \exists f \in F; (S, w, f) \in T^*)$$

Определение 6. *Мы будем писать, что два автомата изоморфны* $(M_1 \cong M_2)$ *тогда и только тогда, когда* $V_1 = V_2$ *и существует биекция* $g : Q_1 \to Q_2$ *такая, что выполняется:*

- $T_2 = \{(g(p), a, g(q)) | (p, a, q) \in T_1\}$,
- $E_2 = \{(g(p), g(q)) | (p, q) \in E_1\}$,
- $S_2 = g(S_1)$,
- $F_2 = \{g(f) | \in F_1\}$.

Определение 7. *Автомат является детерминированным тогда и только тогда, когда*

- $E = \{(q, q) | q \in Q\}$,
- $\forall q \in Q, \forall a \in V$ *множество* $T(q, a)$ *содержит не более одного элемента.*

Для любого автомата можно построить соответствующий ему детерминированный, порождающий тот же язык [1]. Соответствующую операцию над автоматом мы будем изображать как $det(M)$.

Определение 8. *Детерминированный автомат* M *является минимальным, когда любой другой детерминированный автомат, порождающий* $Lang(M)$*, имеет не большее количество элементов в его множестве состояний.*

Для любого детерминированного автомата можно построить соответствующий ему минимизированный. Данную операцию над детерминированным автоматом мы будем изображать как $min(M)$. Для произвольного автомата мы будем писать $min(det(M))$. Равенство языков автоматов означает изоморфность автоматов после операции $min(det(M))$, что дает нам отношение эквивалентности [1]. Соответствующие автоматы мы будем называть эквивалентными.

Определение 9. *Пусть есть два автомата M_1 и M_2, для них выполнено условие $Q_1 \cap Q_2 = \emptyset$, тогда автоматом $M_3 = M_1 \oplus M_2$, порождающим объединение языков автоматов M_1 и M_2, будем считать*

$$(\{q\} \cup Q_1 \cup Q_2, V_1 \cup V_2, T_1 \cup T_2, E_1 \cup E_2 \cup \{(q, S_1), (q, S_2)\}, q, F_1 \cup F_2)$$

При этом $q \notin Q_1 \cup Q_2$.

Условие $Q_1 \cap Q_2 = \emptyset$ накладывается лишь для конструктивности определения и реально не нужно.

Определение 10. *Автоматом M, усеченным по алфавиту W, назовем автомат $M_1 = M|_w$. При этом:*

- $Q_1 = Q$,
- $V_1 = W$,
- $S_1 = S$,
- $F_1 = F$,
- $T_1 = \{(q_1, a, q_2) | (q_1, a, q_2) \in T, a \in W\}$,
- $E_1 = E \cup \{(q_1, q_2) | (q_1, a, q_2) \in T, a \notin W\}$.

Определение 11. *Мы будем говорить, что S-корректный автомат M_1 сильно вкладывается в M_2 начиная с S_2, если*

$$Lang(Q_1, V_1, T_1, E_1, S_1, Q_1) \subseteq Lang(Q_2, V_2, T_2, E_2, S_2, Q_2)$$

Введенное отношение вложения не зависит от F_1, F_2. В контексте исходной задачи это связано с тем, что в автоматах потенциально существуют состояния, в которые можно попасть из начального состояния, но из них нельзя попасть в завершающее состояние. Согласно определению, пути до них не порождают слов автомата. Маркируя все состояния автомата в завершающие, мы насильно добавляем в язык цепочки, порождаемые подобными путями.

Данное определение означает, что при последовательных переходах из начального состояния автомата M_1 мы можем всегда совершить парный переход в автомате M_2, промаркированный тем же символом, что и в автомате M_1.

Мы потребовали S-корректности автомата M_1. Это означает, что при рассмотрении переходов этого автомата у нас не будет недостижимых вершин. То есть мы не рассматриваем конструкцию из множества вершин и отдельной начальной — эта конструкция будет вкладываться куда угодно, но смысла в этом не будет.

Определение 12. *Мы будем говорить, что S-корректный автомат M_1 слабо вкладывается в M_2, если существует $S^* \in Q_2$ такое, что M_1 сильно вкладывается в $(Q_2, V_2, T_2, E_2, S^*, F_2)$.*

Поскольку предыдущее отношение не зависело от F_1, F_2, то данное тоже от них не зависит. Также в данном отношении нет зависимости от S_2. Фактически, мы поочередно выбираем состояния автомата M_2 в качестве начального и проверяем, будет ли выполняться требование сильной вкладываемости начиная с данного начального состояния.

Определение 13. *Мы будем говорить, что M_1 вкладывается в M_2, если:*

- *M_1 — S-корректный автомат,*
- *$M_1|_{V_1 \cap V_2}$ слабо вкладывается в $M_2|_{V_1 \cap V_2}$.*

Соответствующее отношение мы будем обозначать как $M_1 \mapsto M_2$.

В данном отношении нет зависимости от F_1, F_2, S_2; очевидно, что автомат M_1 можно заменить на эквивалентный с тем же алфавитом.

Теорема 1.

$Lang(M_1) \subseteq Lang(M_2) \iff min(det(M_1 \oplus M_2)) \cong min(det(M_2))$

Доказательство.

$$(Lang(M_1) \subseteq Lang(M_2)) \iff$$
$$(Lang(M_1) \cup Lang(M_2) = Lang(M_2)) \iff$$
$$(Lang(M_1 \oplus M_2) = Lang(M_2)) \iff$$
$$(Lang(min(det(M_1 \oplus M_2))) = Lang(min(det(M_2)))) \iff$$
$$(min(det(M_1 \oplus M_2)) \cong min(det(M_2)))$$

□

Последнее введенное определение и является самым главным определением, на основе которого мы будем проверять соответствие SDL-реализации MSC-документации. При этом мы будем пытаться вкладывать автомат, полученный с MSC-диаграмм, в автомат, полученный с SDL-диаграмм. Условие на V-корректность выполняется по построению, поскольку в алфавит мы добавляем только те символы, которые встречаются на диаграммах. Также мы считаем, что условие на S-корректность для MSC также выполняется по алгоритму построения автомата. Недостижимые участки мы выбрасываем сразу. От SDL-автомата мы S-корректности не требуем и это позволяет нам осуществлять проверку даже при незаконченности SDL-диаграмм, когда не все вершины достижимы из начальной диаграммы, но переходы после SDL-состояний являются корректными.

4 Обоснование метода в терминах исходной задачи

Теперь разберемся с самым главным — а что подобная проверка дает с точки зрения исходной задачи. Какими свойствами обладает введенное отношение вложения автоматов в предметной области MSC- и SDL-диаграмм?

Для начала будем использовать следующее приближенное определение ошибки. Если на MSC-диаграммах объекта была специфицирована цепочка **abc** и если на SDL-диаграмме этого же объекта используются символы из алфавита {**a,b,c**}, но нет цепочки $\alpha\mathbf{a}\beta\mathbf{b}\gamma\mathbf{c}\delta$, то это ошибка. При этом $\alpha, \beta, \gamma, \delta$ — некоторые цепочки над алфавитом SDL. Поэтому мы считаем, что такая цепочка обязательно должна существовать.

При выполнении операции усечения автоматов потенциально данная цепочка может преобразоваться в цепочку $\tilde{\alpha}\mathbf{a}\tilde{\beta}\mathbf{b}\tilde{\gamma}\mathbf{c}\tilde{\delta}$. Пусть в контексте задачи символы **a,b,c** обозначают “Вопрос”, “Ответ” и “Подтверждение”. Если мы допустим вставку внутрь цепочки **abc** каких-либо символов из алфавита {**a,b,c**}, то это существенно изменит ее смысл и эта цепочка к первоначальной **abc** будет имет слабое отношение. Поэтому мы делаем предположение, что в рамках предметной области в результате операции

усечения мы получим цепочку вида $\tilde{\alpha}\mathbf{abc}\tilde{\delta}$, т.е. такую, где в цепочку **abc** уже ничего не будет вставлено. Это означает, что в автомате из SDL-диаграммы после операции фильтрации будет существовать цепочка **abc**. При этом она может не начинаться в стартовом состоянии и не заканчиваться в заключительном. Она даже может не являться частью какого-либо слова из языка автомата SDL, например из-за отсутствия в нем заключительных состояний. Но она обязательно существует. Рассмотрением больших цепочек мы получаем, что в контексте предметной области операция усечения имеет смысл и после выбора определенного алфавита сообщений "чистит" автоматы от сообщений, не входящих в данный алфавит так, что поиск цепочек становится более простым. Здесь имеется в виду, что после операции усечения символы цепочки уже идут последовательно.

При реализации и построении алфавитов обоих автоматов для каждого сообщения верен лишь один из трех вариантов.

1. Сообщение принадлежит и SDL-, и MSC-диаграмме. Значение такого сообщения для наших целей уже было разобрано выше. В рассмотренном примере это сообщения из алфавита {**a,b,c**}.

2. Сообщение принадлежит только алфавиту SDL-диаграммы. Здесь надо выдать предупреждающее сообщение, свидетельствующее о том, что SDL реализует какую-то дополнительную функциональность, не специфицированную на MSC.

3. Сообщение принадлежит только алфавиту MSC-диаграммы. Это означает, что SDL-реализация заведомо не может выполнить функциональность, специфицированную на MSC. Здесь также надо выдать соответствующее предупреждающее сообщение о возможной ошибке (хотя такая ситуация может быть допустимой, если одна MSC-роль была разбита на несколько SDL-сервисов).

Следующей операцией, которую мы производим в процессе проверки, является операция перемаркировки всех состояний автоматов в завершающие. Очевидно, что на возможность найти одну цепочку в другом автомате данная операция не влияет. Операция перемаркировки состояний является частью преобра-

зования автоматов для того, чтобы не потерять никакие цепочки. Например, при создании MSC-диаграмм основное внимание уделяется специфицированию обмена сообщениями. При этом символ завершения объекта часто опускается, а также может подразумеваться наличие некой функциональности после данного сценария. Это приводит к тому, что язык порождаемых автоматов может оказаться пустым из-за отсутствия символа завершения. А нам надо сохранить те цепочки, которые на нем специфицированы. Поэтому перевод всех состояний в завершающие — дань предметной области.

Обсудим еще один тонкий момент — переход от нахождения существования отдельных цепочек к вкладываемости автоматов. В терминах исходной задачи это выражено в том, что если на MSC определен некий связный блок поведения, то и на SDL ему должен соответствовать связный блок. В рамках рассматриваемого подхода мы считаем, что интуитивное понятие связности, если SDL-реализация соответствует MSC-документации, не должно изменяться. Рассмотрим гипотетический пример. Если на MSC-диаграмме сначала идет общая преамбула, а затем есть четыре варианта поведения, то и на SDL-диаграмме должно быть нечто, похожее на преамбулу и четыре варианта. А если там будет реализована преамбула, а затем будет лишь три варианта, а четвертый будет реализован где-то в стороне, то данный подход будет считать это несоответствием, хотя в этом примере цепочки, соответствующие всем четырем вариантам, будут успешно найдены. Поэтому мы требуем, чтобы найденные цепочки имели общее начало. Это реализуется поиском состояния, куда вкладывается целый автомат с MSC-диаграммы. Вкладываемость автомата, согласно данным определениям, соответствует вкладываемости языка, порождаемого автоматом, а это сразу обеспечивает то, что все цепочки языка будут иметь общее начало.

Согласно примечанию к определению вкладываемости, нет зависимости от начального состояния S_2. В качестве этого состояния практически выступает образ стартового состояния с SDL-диаграммы. В терминах исходной задачи это достаточно естественно.Искомая функциональность должна где-то суще-

ствовать, но ее выполнение может не начинаться с самого начала SDL-диаграммы — может не следовать сразу за стартовым состоянием.

Мы последовательно рассмотрели все операции, которые мы производим в процессе проверки вкладываемости одного автомата в другой. Подведем итоги обсуждения. Метод может осуществлять проверку в случаях, которые соответствуют естественному развитию системы в течение жизненного цикла. Приведем пример.

Пусть на этапе проектирования был создан ряд MSC-диаграмм. Завершения функциональности объектов на них никто не указывал, поскольку планировалось, что на данной группе сценариев объект не заканчивает свое существование. Потом была построена SDL-модель, реализующая данную функциональность. На SDL-модель позже были добавлены дополнительные обмены сигналами. Внутрь нее — сигналы для сценариев техобслуживания. Также сигналы были добавлены до и после отработки данной функциональности. Еще SDL-модель была преобразована таким образом, что объект стал заканчивать свое существование после ее отработки. Над первоначально изображенной схемой на MSC-диаграмах на SDL-модели были проведены изменения, позволяющие ей выполняться циклично.

Все остальные методы проверки будут выдавать ошибку, поскольку SDL-диаграмма была изменена слишком сильно относительно исходной MSC-документации. Это естественно, так как соответствия здесь уже нет. Но можно проверить, что между MSC-документацией и SDL-реализацией нет противоречий. Это обеспечивается предлагаемым методом. Фактически, он может “найти” исходную модель внутри преобразованной при условии, что она удовлетворяет вышеприведенному набору предположений. В описанном случае он подтвердит, что противоречий нет. В другом случае, если как-либо изменить образ самой первой SDL-модели, построенной по MSC в преобразованном SDL, то метод справедливо выявит ситуацию ошибки.

5 Реализация

Для начала полностью выпишем алгоритм, проверяющий соответствие SDL-реализации MSC-документации для одной сущности, а затем прокомментируем, на основе чего взялись те или иные пункты. В квадратных скобочках даны необязательные улучшения алгоритма, позволяющие увеличить его скорость работы.

1. По проверяемой паре MSC-SDL создаются конечные автоматы.
   Под MSC имеется в виду MSC-диаграмма, выбранная в качестве начальной, и все те, которые оказались с ней связаны. См. раздел 6.

2. Все состояния автоматов перемаркируются в завершающие.

3. Анализируются списки сообщений, выдаются соответствующие предупреждения, выполняется процедура усечения автоматов по списку сообщений, принадлежащим обоим автоматам.

4. Запускается алгоритм проверки вложения усеченного автомата из MSC в усеченный автомат из SDL.

   (a) [Детерминизировать и минимизировать усеченный автомат из MSC.]
   Результат обозначим как $A - MSC$ [1].

   (b) По всем вершинам из усеченного SDL-автомата.

      i. Считать начальной вершиной усеченного SDL-автомата выбранную.

      ii. [Проверить возможность вложения $A - MSC$ по начальным символам слов в новый SDL-автомат. Перейти к следующей вершине при определении невозможности вложения.]

      iii. Детерминизировать и минимизировать новый SDL-автомат. Результат обозначим как $A - SDL$.

      iv. Склеить автоматы $A - MSC$ с $A - SDL$ согласно Определению 9.

[1] Если операция детерминизации и минимизации не проводилась, то вместо данного автомата мы берем усеченный MSC-автомат.

v. Детерминизировать и минимизировать автомат с предыдущего шага. Результат обозначим как $A-\sum$.

vi. Если $A-\sum$ изоморфен $A-SDL$, то внести данную вершину в множество вершин, начиная с которых усеченный MSC-автомат вкладывается в усеченный SDL-автомат.

(c) Выдать вершины, начиная с которых усеченный автомат из MSC вкладывается в усеченный автомат из SDL.

Определения вложения были даны неконструктивно, но приведенная теорема позволяет построить конструктивный алгоритм проверки вложения.

Пройдемся по дополнительным пунктам, которые мы ввели:

1. усеченный автомат из MSC-диаграмм можно минимизировать, поскольку от замены данного автомата на эквивалентный порождаемый язык не меняется. Это дает выигрыш в последующих алгоритмах, сложность которых зависит от количества вершин;

2. следует более осторожно пользоваться алгоритмами $min(det(\dots))$, имеющими большую сложность. Нам требуется проверить, что $Lang(M_1) \subseteq Lang(M_2)$ после того, как все состояния были перемаркированы в завершающие. Пусть нам известно, что слова длины n образуют множества P_1 и P_2 соответственно, и для них известно, что $P_1 \not\subset P_2$. Тогда из этого сразу следует, что $Lang(M_1) \not\subset Lang(M_2)$. А n можно взять равным 1 или 2.

Теперь о сложности используемых алгоритмов. Верхние границы временной сложности для алгоритмов по количеству вершин автоматов:

- детерминизации — экспоненциальный;
- минимизации — $O(n * ln(n))$;
- проверки на изоморфность — $O(n^2)$.

На практике верхние границы являются завышенными:

- для алгоритма детерминизации автоматы берутся из MSC- и SDL-диаграмм. Это означает, что соединений между вершинами мало и итеративный алгоритм детерминизации сходится быстро;
- от связки детерминизация-минимизация часто можно избавиться с помощью п. 2, проверяя только те вершины, для которых вложение может состояться;
- в алгоритме изоморфности автоматы могут отсеяться до основной проверки, например, по количеству вершин и по количеству вершин в множестве заключительных состояний.

Это и дает возможность практического применения. Дальнейшие возможности по увеличению быстродействия уже стоит обсуждать в контексте конкретной реализации конечных автоматов.

При проверке системы мы последовательно рассматриваем все возможные MSC-SDL пары. Для данных пар должно выполняться условие наличия на них общих сигналов. Таким образом производится проверка всей системы, а затем пользователь может просмотреть результаты анализа.

6 Выбор начальной MSC-диаграммы

По множеству базовых MSC-диаграмм[2] для каждого объекта можно выделить, как его диаграммы связаны друг с другом с помощью reference и inline конструкций. Вся совокупность MSC-диаграмм объекта образует несвязный ациклический направленный граф. При построении конечного автомата нам надо указать, с какой диаграммы его надо начать строить. Автоматически определить, какая диаграмма является “начальной”, невозможно. Поэтому она задается. После этого на множестве MSC-диаграмм организуется порядок и мы получаем дерево, начинающееся с выбранной MSC-диаграммы.

[2]Для HMSC[8] такой проблемы не возникает. Это еще раз подчеркивает ценность данного метода, поскольку он применим к уже созданной документации на MSC-диаграммах. Существующие диаграммы не нужно приводить к HMSC описанию, чтобы получить результат.

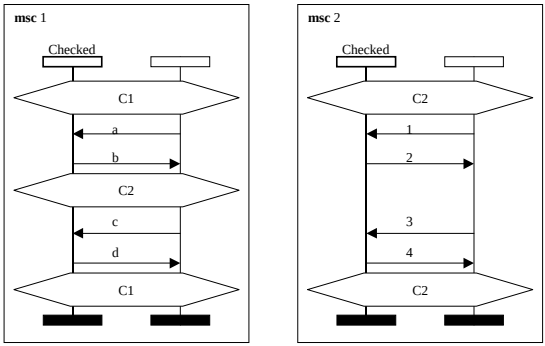

Рис. 3: MSC-диаграммы 1, 2 для проверки соответствия

В зависимости от наличия ошибок в соответствии между MSC- и SDL-диаграммами выбор начальной диаграммы может влиять на результат проверки:

- при “соответствии” SDL-диаграмм множеству MSC-диаграмм данного объекта метод будет давать положительные результаты для любой MSC-диаграммы, выбранной в качестве начальной;
- при наличии ошибок в “соответствии” для каких-то диаграмм, выбранных в качестве начальной, ответ может быть положительным, а для каких-то — отрицательным.

В качестве пояснения можно рассмотреть пример, когда у нас есть три *MSC-диаграммы* A, B, C. A порождает символ a и сслылается на диаграмму B, B порождает b и ссылается на C, которая порождает c. Если мы выберем в качестве начальной диаграммы A, то автомат будет порождать цепочку abc. Если B — bc, если C, то c. Далее надо рассмотреть SDL-диаграмму, порождающую $abbc$.

При организации проверки всей системы метод позволяет рассчитать результаты проверки для всех MSC-диаграмм в качестве начальной и выдать интерпретацию результата.

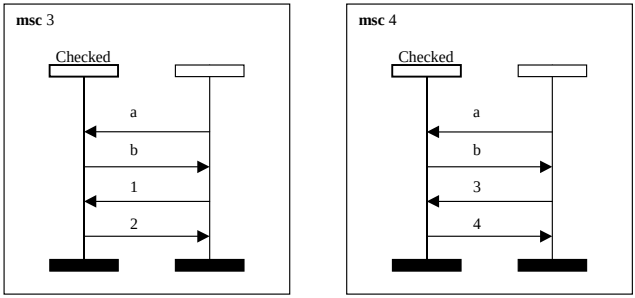

Рис. 4: MSC-диаграммы 3, 4 для проверки соответствия

7 Пример

Пусть существуют MSC-спецификации 1, 2, 3, 4. Спецификации 1 и 2 изображены на рисунке 3, а спецификации 3 и 4 — на рисунке 4. Пусть также дана SDL-реализация, приведенная на рисунке 5. Ставится целью проанализировать соответствие реализации различным наборам спецификаций.

Наш метод позволяет выявить, что SDL-реализация:

1. удовлетоворяет MSC-спецификации 1, без учета спецификаций 2, 3 и 4;
2. удовлетворяет MSC-спецификации 2, без учета спецификаций 1, 3 и 4;
3. удовлетворяет спецификациям 1 и 2 вместе, если начальной является спецификация 1;
4. удовлетворяет спецификациям 1 и 2 вместе, если начальной является спецификация 2;
5. удовлетворяет спецификации 3;
6. не удовлетворяет спецификации 4.

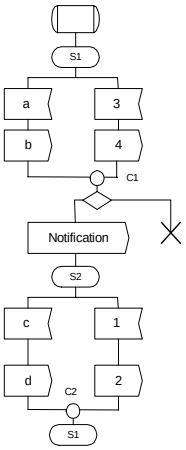

Рис. 5: Проверяемая SDL-диаграмма

8 Возможные усовершенствования и дальнейшее развитие метода

В данной статье описаны базовые идеи по проверке соответствия SDL-реализации MSC-документации. В дальнейшем метод планируется расширять. Мы приведем некоторые идеи, которые могут как сделать его более удобным в применении, так и перерасти в отдельную область по проверке соответствия SDL- и MSC-моделей, базирующуюся на конечных автоматах. Ожидается, что данная область будет дополнять существующие подходы на основе трасс и состояний протокола.

1. Разрешить пользователю редактировать словари сообщений, по которым сравниваются MSC- и SDL-диаграммы.

2. Предоставлять возможность задавать множество MSC-диаграмм, используемых для проверки. При этом можно будет исключить из проверки диаграммы, являющиеся источником проблем.

3. На текущий момент данный метод выдает достаточно жесткое решение — либо да, либо нет. В текущей реализации пользователю приходится самостоятельно классическими

средствами анализа трасс искать, где находится точка расхождения. Данный процесс можно автоматизировать для нахождения ближайшей точки расхождения диаграмм.

4. С помощью конечно-автоматных моделей можно также определить, насколько SDL-реализация отличается от MSC-спецификации. Это можно сделать на основе построения разности языков. Разность конечных автоматов из MSC- и SDL-диаграмм будет давать автомат, порождающий разность языков. При необходимости на основе него можно как построить наборы слов данного языка, соответствующие трассам сигналов, так и представить его в виде SDL-диаграмм с помощью методов, используемых в работах [13, 12]. Особую ценность это имеет, когда ясно, что SDL-реализация соответствует MSC-спецификации, но не ясно, а что же она делает еще. Ни одно из известных автору средств не предоставляет подобной информации.

5. Идею предыдущего пункта можно развернуть и предоставлять информацию о покрытии MSC-спецификациями SDL-реализации. Этого тоже нет ни в одном средстве, и наиболее ценно применение данного пункта — в тестировании. Отличие от покрытия с помощью трасс в том, что здесь можно будет задавать "тесты" не в виде конечного числа событий (трассы), а в виде сценариев с циклами. Если предыдущий пункт следует применять, когда существует единое описание объекта на MSC-диаграммах, то данный пункт применим, когда есть много разрозненных MSC-диаграмм, описывающих части SDL-поведения.

9 Место работы среди существующих методов

С точки зрения автора, круг подходов по верификации SDL-диаграмм по MSC-диаграммам практически всегда ограничивается рассмотрением трасс, с возможным приложением к тестированию, и рассмотрением состояний протокола [7]. Даже в стандартах серии ETSI [5, 3, 4] упомянуты именно эти два подхода. Таким образом, средство Telelogic Tau [17] действитель-

но соответствует стандартам и поддерживает все существующие подходы.

В работе [16] упоминалось о возможности пропускать сообщения на проверяемой модели. Но поскольку в ней рассматриваются только отдельные трассы, то достоверность такого подхода невелика. Автору не известны средства, реализующие подобный подход. В качестве аннотации существующих средств использовалась работа из области, близкой к данной [15].

В смежной области формальных методов существует отдельный подход по сравнению конечно-автоматных моделей на основе вложения языков. Хорошее описание формальных методов можно найти в книге NASA [6]. Использование конечно-автоматных моделей без словарей сообщений [11] будет похоже на алгоритмы [7] с той только разницей, что используются не переборные методы, а формальные. Соответственно, их использование будет иметь те же недостатки, что и средство Telelogic Tau.

Предлагаемая работа выигрывает именно за счет интеграции нескольких подходов:

- переход от SDL и MSC к конечно-автоматным моделям с учетом специфики SDL и MSC;
- операции фильтрации, позволяющей разделить общее множество сигналов на “используемые” и “неиспользуемые”;
- проверки на включение языков, реализуемой с помощью конечных автоматов.

Это дало возможность создать модель, в рамках которой можно проверять непротиворечивость диаграмм даже тогда, когда известно, что диаграммы не полностью соответствуют друг другу. Данный подход можно использовать как дополнение к существующим, когда их возможностей уже не хватает.

Список литературы

[1] Рейуорд-Смит В. Дж. Теория формальных языков. — М.: Радио и связь, 1988. — 128 с.

[2] Сервер группы OMG, занимающейся развитием методологии UML. Стандарты, публикации, средства. — http://www.omg.com.

[3] EG 201 015 Ver. 1.2.1, Methods for Testing and Specification (MTS); Specification of protocols and services; Validation methodology for standards using Specification and Description Language (SDL); Handbook, 1999.

[4] ETS 300 414, Methods for Testing and Specification (MTS); Use of SDL in European Telecommunication Standards Rules for testability and facilitating validation, 1999.

[5] [3]ETR 184 : Methods for Testing and Specification (MTS); Overview of validation techniques for European Telecommunication Standards (ETSs) containing SDL, 1995.

[6] Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems, Volume II: A Practitioner's Companion. — 1997. — NASA-GB-001-97. — 245 p.

[7] Holzmann G. Design and Validation of Computer Protocols — Prentice-Hall: 1991 — 512 p.

[8] ITU-T MSC2000R3 Draft Z.120: Message Sequence Charts ITU-T Recommendation Z.120. — November 1999.

[9] ITU Recommendation Z.100: CCITT SPECIFICATION AND DESCRIPTION LANGUAGE (SDL). — 1993. — 272 p.

[10] ITU Recommendation Z.120: Message Sequence Chart (MSC). — 1993. — 91 p.

[11] Kurshan R. Computer-Aided Verification of Coordinating Processes // Princeton Series in Computer Science. — 1994.

[12] Mansurov N. Automatic Synthesis of SDL from MSC in Forward and Reverse Engineering // Proc. of the 1st International Conference on Visual Formal Methods. — 1999. — P. 44-64.

[3]Текущий статус данных документов можно узнать на сайте www.etsi.org

[13] Mansurov N., Zhukov D. Automatic synthesis of SDL models in Use Case Methodology // Proc. of the 9th SDL Forum. — 1999. — P. 225-240

[14] Rumbaugh J., Jacobson I., Booch G. The Unified Modeling Language Reference Manual. — Boston: Addison-Wesley, 1999. — 576 p.

[15] Schmid R., Ryser J., Berner S., Glinz M., Reutemann R., Fahr E. A Survey of Simulation Tools for Requirements Engineering: Technical Report 2000.06 — Zurich, Institut für Informatik,: 2000.

[16] Sinclair D., Stone B., Clynch G. An Object Oriented Methodology from Requirements to Validation // Proc. of the 2nd Object Oriented Information Systems Conference. — 1995. — P. 265-286.

[17] Telelogic Tau 4.2 documentation, 2001.