ПРОГРАММИРОВАНИЕ, 2007, № 1, с. 35–50

УДК 004.4′22 + 004.415.2 + 519.687.4 + 004.434

РЕАЛИЗАЦИЯ СТЫКА МЕЖДУ MSC- И SDL-ДИАГРАММАМИ В ТЕХНОЛОГИИ REAL

© **2007 г. А. Н. Терехов, В. В. Соколов**

Санкт-Петербургский государственный университет

198504 Санкт-Петербург, Петродворец, Университетский просп., 28

E-mail: ant@tercom.ru, svv@tercom.ru

Поступила в редакцию 13.10.2004 г.

Рассматриваются различные методы стыковки MSC- и SDL-диаграмм, в том числе позволяющие автоматизировать ручной переход от сценариев поведения системы в целом (HMSC/MSC) к поведенческой модели каждого объекта (SDL), методы поддержки их адекватности в течение всего жизненного цикла – согласованное внесение изменений, расширения MSC-диаграмм для реальных ситуаций. Приводится обзор существующих алгоритмов, рассматриваются их сильные и слабые места, предлагаются собственные подходы, устраняющие недостатки. Предложенные алгоритмы объединяются в единый комплекс.

1. ВВЕДЕНИЕ

Наш коллектив занимается разработкой ПО для телефонных станций и других систем реального времени с начала 80-х годов. Оказалось, что самым трудным моментом является взаимодействие заказчиков, алгоритмистов, программистов, специалистов по протоколам, электронщиков и т.д. Причем проблемы возникают не из-за недостаточной квалификации кого-либо из участников разработки, а из-за взаимного недопонимания. Для преодоления этих трудностей Международный Консультационный Комитет по Телеграфии и Телефонии (МККТТ, ныне ITU-T) разработал серию графических способов описания, в частности, языки SDL [1] и MSC [2].

SDL-диаграмма (SDL – Specification and Description Language) очень похожа на традиционные блок-схемы, но с несколькими важными расширениями: символ состояния, в котором процесс не занимает процессор, ожидая приема одного или нескольких сигналов; символ приема сигнала и символ посылки сигнала, как это изображено на рис. 1. В SDL-диаграммах основное внимание уделяется логике использования сигналов; логика выполнения действий, не связанных с сигналами, детализируется лишь по мере необходимости. Поэтому действие может быть и атомарным, как показано на рис. 1, и достаточно сложным, состоящим из десятков или сотен операторов.

MSC-диаграммы[1] (MSC – Message Sequence Chart) позволяют описывать сценарии поведения системы во времени. Время течет сверху вниз, вертикальные линии представляют объекты системы, а между ними рисуются стрелки, обозначающие сигналы[2]. Элементарная MSC-диаграмма изображена на рис. 2. MSC-диаграммы состоят из отдельных сценариев и структур, задающих последовательности выполнения этих сценариев. Они широко применяются для описания протоколов различными международными организациями, в том числе для стандартов, разработанных организацией ITU-T, однако редко когда приводятся исчерпывающие описания поведения системы, включая обработку аварийных ситуаций, техобслуживания, тарификации и т.д. И MSC-, и SDL-диаграммы применяются для

[1] Сейчас все большее распространение получают HMSC-диаграммы [3], которые являются развитием MSC-диаграмм. По своей сути они во многом остаются такими же, поэтому отдельно мы их не рассматриваем и термином MSC обозначаем и их тоже.

[2] В данной статье мы считаем понятие SDL-сигнала взаимозаменяемым с понятием MSC-сообщения.

Рис. 1. Пример SDL-диаграммы.

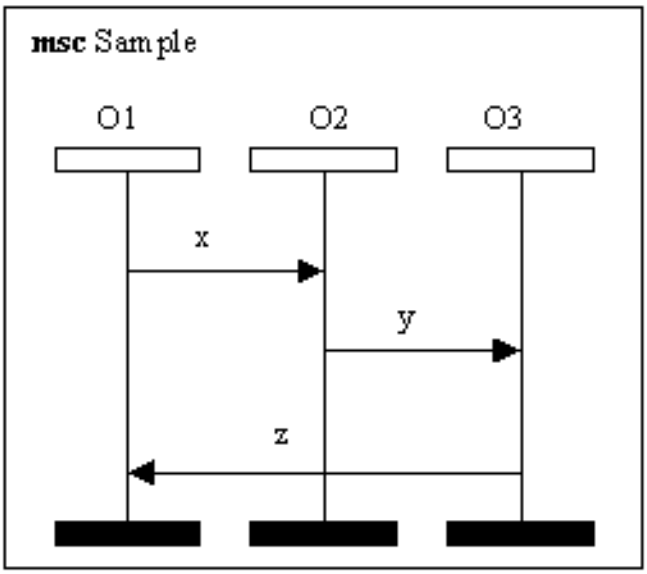

Рис. 2. Пример MSC-диаграммы.

описания динамического[3] поведения системы, их сравнительный анализ приведен в таблице.

В современных системах используются оба стандарта, поскольку они дают различную информацию о динамическом поведении объекта и дополняют друг друга. Точно так же они оба нужны с технологической точки зрения. Соот-

[3] Данные модели имеют средства описания и статической части системы тоже, но проблем с этим гораздо меньше.

Сравнительные характеристики SDL- и MSC-диаграмм

MSC-диаграммы	SDL-диаграммы
Описывают взаимодействие между различными частями системы; по отношению к объекту являются внешними	Описывают внутренние алгоритмы каждого из объектов; по отношению к объекту диаграммы являются внутренними
Отображена логика взаимодействия нескольких объектов. Логика поведения каждого из них скрыта	Отображена логика поведения каждого из объектов. Логика их взаимодействия скрыта
Информация о выборе линии поведения дается словесно	Логика работы детализована вплоть до значений переменных
Создаются на этапе проектирования	Создаются при программировании
Используются при обсуждении задачи с экспертами предметной области	Преимущественно используются программистами

ветственно, возникает необходимость обеспечения их согласованности.

В данной статье мы кратко обосновываем, почему в одном и том же средстве используются и MSC-, и SDL-диаграммы, описываем существующие подходы по их одновременному использованию и указываем на проблематичные места. Наше технологическое средство REAL [4] также использует и MSC-, и SDL-диаграммы, и мы описываем наш подход к разрешению проблем.

2. ОБСУЖДЕНИЕ ОБЛАСТИ

Мы уже пояснили, что MSC и SDL дают различные взгляды на систему. Подчеркнем, что они являются не просто различными представлениями одной и той же модели, а несут в себе разную информацию. Например, только из SDL-диаграмм можно узнать о выборе одной либо другой ветви поведения. Соотвествующая информация содержится в переменных, которые в MSC-диаграммах отсутствуют.

Для того, чтобы указать информацию, которая содержится только в MSC-диаграммах, рассмотрим гипотетическую ситуацию с банком, когда для некоторых клиентов кассир сразу выдает деньги, а для других – предварительно консультируется у директора банка. В общем случае, имея SDL-модели всех участников, нельзя восстановить, участвовал ли директор в процедуре выдачи денег конкретному клиенту. Можно показать, что эта задача родственна задаче о самоприменимости и алгоритмически неразрешима. Соответствующая информация хранится в MSC-диаграммах, которые содержат информацию не только о том, как объекты обмениваются сообщениями, но и кто участвует в сценариях взаимодействия.

Поэтому за какой-то информацией приходится обращаться к MSC-диаграммам, а за какой-то – к SDL-диаграммам. Из-за этого в современных технологических средствах присутствуют оба типа диаграмм. На рис. 3 изображен процесс создания системы в нашем технологическом средстве REAL на момент написания статьи [4]. Видно, что между MSC- и SDL-моделями существует разрыв, который замедляет скорость разработки из-за двойного создания динамики системы сначала на MSC, потом на SDL, и просто может служить источником ошибок при несогласованности моделей. С точки зрения авторов, в других подходах, таких, как в UML с Sequence- и Collaboration-диаграммами [5, 6], так и в подходах на базе MSC без UML, или в подходах на основе UseCaseMaps [7], также возникает разрыв между MSC-подобной моделью и SDL.

Для технолога было бы идеальным вариантом, если бы существовала некая объемлющая модель, на которую, как на трехмерный кубик,

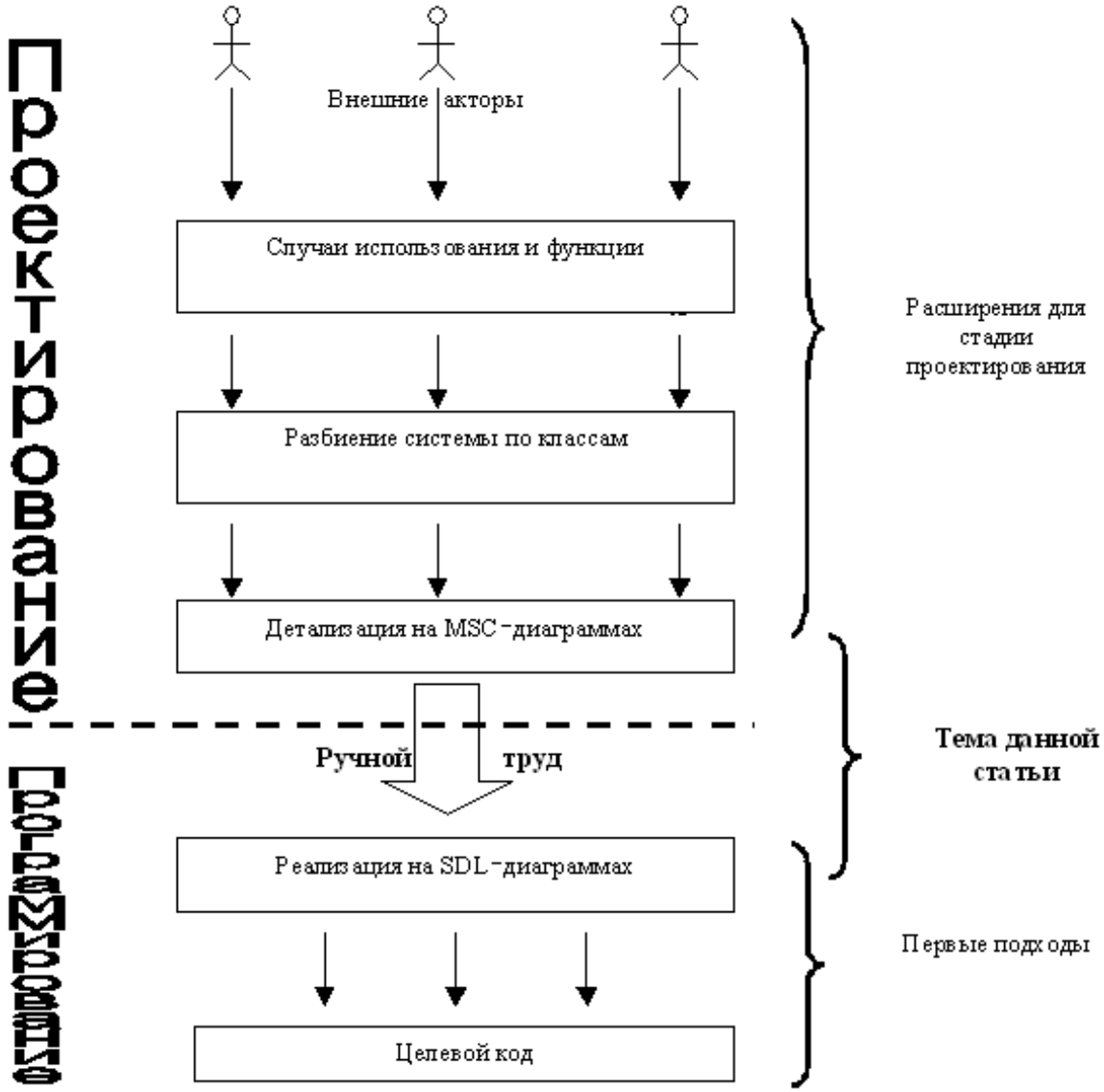

Рис. 3. Место SDL- и MSC-моделей в технологии REAL на момент написания статьи [4].

можно было бы взглянуть с одной стороны и увидеть SDL-диаграммы, а с другой грани она бы давала представление в виде MSC-диаграмм. Авторам не известно о существовании подобной модели, и, на текущий момент времени, они могут выделить лишь ряд подходов по согласованию MSC- и SDL-диаграмм в существующих CASE-системах. Условно их можно разделить на следующие группы.

1. MSC и SDL независимы.
   (a) MSC и SDL представляют собой отдельные независимые типы диаграмм. В этом случае их несогласованность ведет к потенциальным ошибкам.
   (b) Переход от MSC-диаграмм к SDL-диаграммам осуществляется человеком, что вносит ошибки и очень часто приводит к неоптимальным решениям, лишнему и избыточному коду, особенно при длительной поддержке проекта.
2. Попытки создавать системы на основе MSC-диаграмм с последующим синтезом SDL.
   (a) В случае, когда пытаются поддерживать весь цикл разработки, это влечет

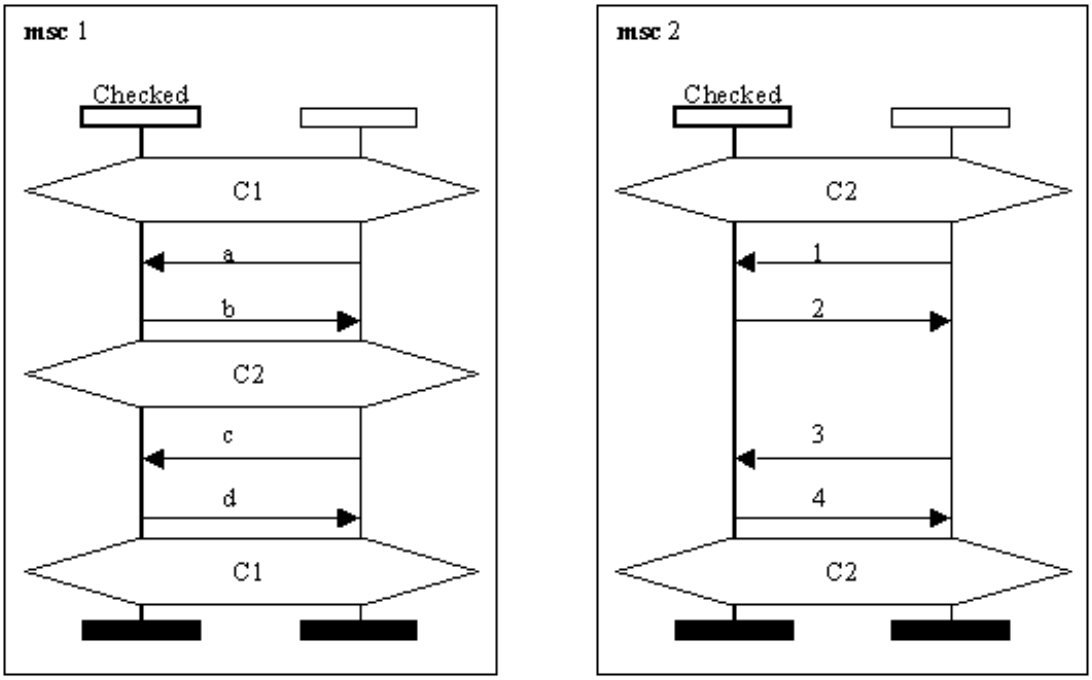

Рис. 4. MSC-диаграммы 1, 2 для проверки соответствия.

за собой расширение стандарта MSC и перенос в MSC-диаграммы кода, данных и других технических деталей [8]. После этого начинаются проблемы при увеличении количества сценариев и вытекающей нестыковке участков кода. Структура MSC малоприменима для задания на ней кода. MSC-диаграммы больше отвечают на вопрос "Что может случиться?", чем на вопрос "Почему так случается?", вследствие чего они обладают избыточностью и мало приспособлены для использования в них "арифметических" данных.

(b) "Односторонний синтез". В этом случае один раз производится синтез SDL-диаграмм по MSC-диаграммам, а затем используются полученные структуры [8–10]. В таком случае внесение изменений вручную очень дорого стоит. Зачастую это приводит к полному отказу от MSC-диаграмм, по которым был проведен синтез.

(c) "Инкрементальные алгоритмы". Стандарт MSC урезается до диаграмм, задающих только трассы[4]. Существуют алгоритмы [11], позволяющие добавить новую MSC-спецификацию – трассу в SDL-диаграммы. Данный подход работает лишь на специфическом классе задач из-за постоянного присутствия цикличности в реальных задачах. Для неурезанного стандарта MSC задача создания инкрементальных алгоритмов также рассматривалась, но была разрешена лишь для очень узкого набора изменений [12].

Алгоритмы, используемые в пунктах 2.c и 2.b [9, 10], способны провести синтез не из любых MSC-описаний. Алгоритм пункта 2.a способен на это всегда. Побочным эффектом этого алгоритма является то, что в процессе его работы данные могут быть преобразованы таким образом, что в получившейся SDL-диаграмме будет тяжело узнать исходную MSC-программу. Сразу отметим, что невозможность осуществить синтез во многих разумных ситуациях яв-

[4] В данной статье под трассой мы понимаем цепочки сообщений.

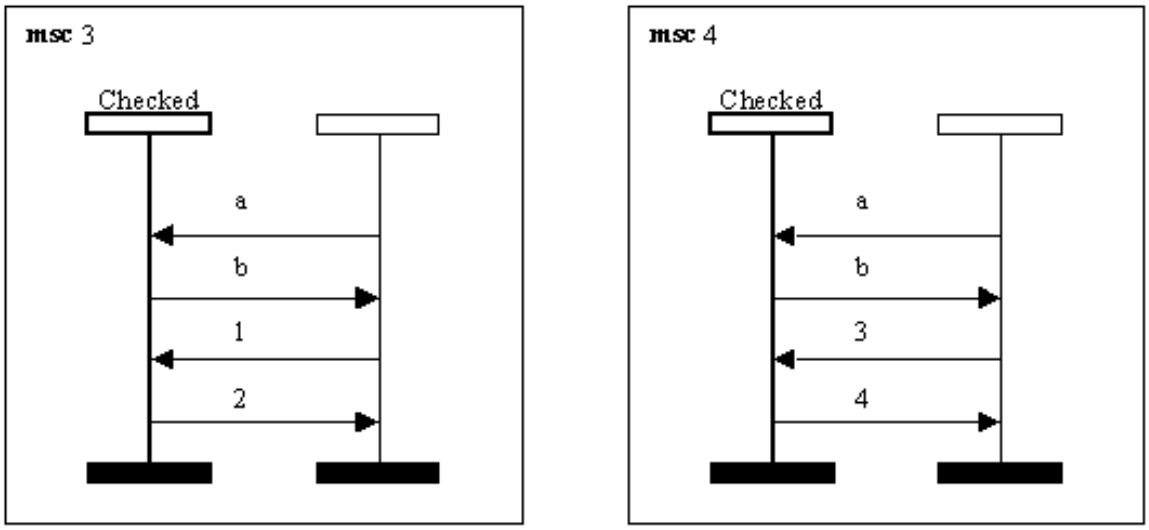

Рис. 5. MSC-диаграммы 3, 4 для проверки соответствия.

ляется очень слабым местом, поэтому авторы отдают предпочтение вариантам алгоритма 2.a.

3. Алгоритмы сравнения используемых MSC- и SDL-диаграмм для проверки соответствия.

   (a) На основе MSC-диаграмм строятся трассы, а затем полученные трассы "накладываются"[5] на SDL-диаграммы [13–15]. Если трасса укладывается, то все хорошо. Если нет – это считается ошибочной ситуацией. Подобный способ недостаточно эффективен, поскольку добавление где-либо отладочного сообщения нарушает последовательность сигналов на трассе, и считается, что SDL-модель не соответствует MSC-спецификациям [16]. Этим методом невозможно проверить случай, когда SDL-модель является реализацией сразу нескольких ролей из MSC-диаграмм.

   (b) Авторам встречалась идея модификации предыдущего алгоритма с разрешением пропускать сигналы. Она была описана в [17], но широкого распространения не получила из-за невысокой достоверности результата.

   (c) Использование одного из алгоритмов [18], связанных с анализом внутренних состояний протокола, при этом параллельно выполняются две модели – MSC и SDL. В процессе выполнения проверяется, что принятые и посланные сигналы одинаковы. Данные алгоритмы изначально разрабатывались для проверки непротиворечивости моделей и перестают работать для проверки соответствия SDL-диаграмм MSC при наличии разногласий между MSC и SDL, типа отсутствия цикличности в MSC-документации и ее присутствия в SDL-модели.

С точки зрения авторов, в CASE-системе должны присутствовать как MSC-, так и SDL-диаграммы для возможности создания нескольких различных взглядов на динамику системы. Практика показывает, что в промышленных системах SDL должна быть не только "документацией", но и "программой", по которой можно автоматически породить исполняемый код. Для построения интегрированной системы для MSC- и SDL-диаграмм следует использовать как процедуры перехода от MSC к SDL – генерацию, так и обратно – средства верификации. При

[5] Имеется ввиду посимвольное совпадение сигналов трассы с сигналами диаграммы.

этом следует выделить следующие проблемные места, которые следует доработать:

- улучшить процедуру верификации так, чтобы она позволяла более надежно проверять соответствие моделей при их рассогласовании, что неизбежно в течение жизненного цикла;
- изменение алгоритма генерации [8], чтобы на него можно было повлиять для получения более наглядной и удобной SDL-модели, а не единственного варианта, выдаваемого генерацией;
- улучшить описательные свойства MSC, что даст возможность создавать более полные модели MSC уровня. Это нужно и для генерации, и для верификации, и просто для более детальной проработки спецификаций.

Наиболее сильной связкой, которая присутствует сейчас на рынке, является пара Teleleogic Tau [16] и средства генерации диаграмм KLOCWork[6] [20, 21]. Это интегрированный подход, в котором используются и генерация, и верификация (алгоритмы 3.c, 3.a, 2.b по нашей классификации). Данная связка не свободна от поставленых проблем.

3. ПРЕДЛОЖЕННЫЕ РЕШЕНИЯ

На базе технологического средства [4] нами был разработан ряд алгоритмов, решающих проблемы, поставленные в разделе 2. Данное средство было дополнено следующими нововведениями:

- расширенный язык MSC для создания более полных MSC описаний системы;
- настраиваемая генерация из MSC-модели в SDL-модель;
- собственная процедура проверки соответствия SDL-программы MSC-документации.

3.1. Новый способ верификации

Разработан специальный способ верификации для проверки соответствия SDL-диаграмм MSC-документации в случае, когда между MSC- и SDL-диаграммами есть различия, но известно, что оба типа диаграмм соответствуют поведению одного объекта. В отличие от остальных методов, которые базируются на различных доказательствах того, что SDL-модель “такая же”, как и MSC-модель, наш метод базируется на понятии инварианта, который сохраняется при определенном классе изменений MSC- и SDL-моделей в жизненном цикле разработки. В нем MSC- и SDL-модели могут отличаться и при этом соответствовать друг другу, но только до того момента, пока они сохраняют общий инвариант. В качестве очень упрощенного примера считается, что MSC-модель, задающая трассу *abc*, соответствует SDL-модели, задающей трассу *abecd*, но не соответствует SDL-модели *abbc*. Для объяснения выбранного реше-

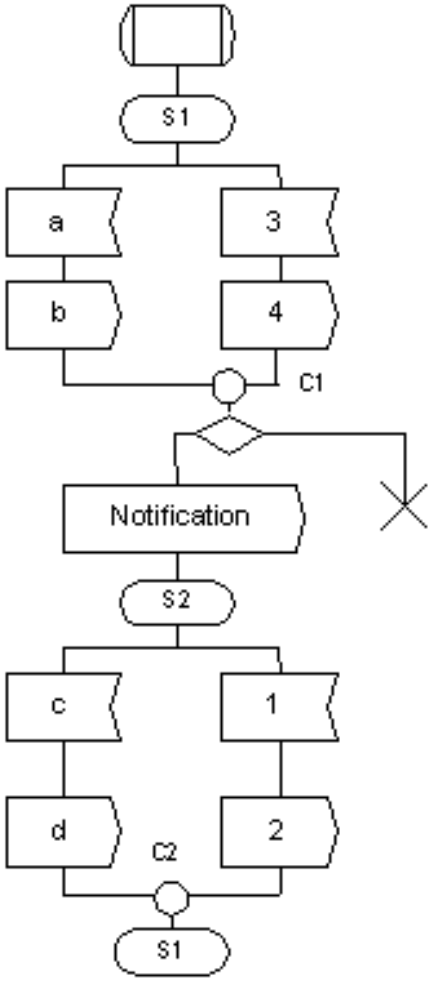

Рис. 6. Проверяемая SDL-диаграмма.

[6] Является официальным партнером Telelogic; часть этого проекта раньше была известна под названием MOST [19].

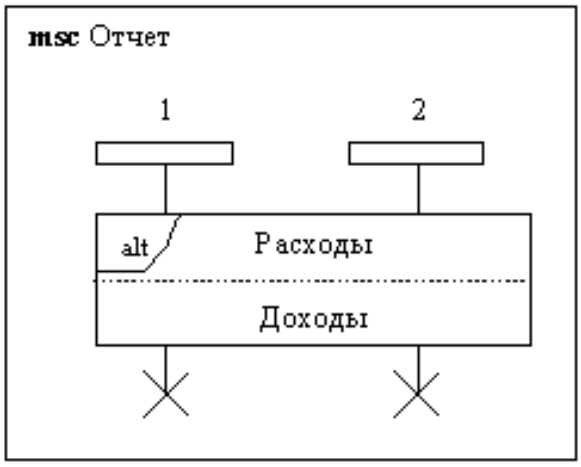

Рис. 7. Главная MSC-диаграмма для примера генерации.

ния пусть символы a, b, c обозначают "Вопрос", "Ответ", "Подтверждение". Тогда первая SDL-модель характеризуется реализацией какой-то дополнительной функциональности, не нарушающей исходную, а вторая модель задает совсем другой алгоритм обмена.

При проверке MSC–SDL мы оставляем в обеих моделях только символы из множества $\{a, b, c\}$. Это множество содержит все общие символы двух моделей. При этом (усеченная) MSC-трасса будет накладываться на усеченную первую SDL-модель, но не будет укладываться на усеченную вторую. В реальном алгоритме проверки требуется, чтобы "накладывался" весь конечный автомат из MSC-диаграмм. Это требование соответствует существованию наложения для каждой трассы, причем все трассы должны начинаться в одной точке, а не просто существовать где-либо в SDL-диаграмме.

Алгоритм проверки[7]

1. Последовательно по всем проверяемым объектам SDL-диаграмм:

2. Выбрать SDL-диаграмму данного объекта и множество проверяемых MSC-диаграмм, в которых присутствует данный объект.

3. Последовательно по всем проверяемым MSC-диаграммам задаем их в качестве начальной.

4. Строим конечные автоматы для обоих типов диаграмм. Для SDL в качестве начального состояния берется начальное SDL-состояние, для MSC – диаграмма, выбранная в качестве начальной в пункте 3.

5. Каждый из конечных автоматов должен порождать множество трасс выбранного объекта с точки зрения данного типа диаграмм. Поэтому все состояния конечных автоматов перемаркируются в завершающие.

6. Задается множество "существенных" сообщений – тех сообщений, которые присутствуют в каждом из конечных автоматов.

7. Заменить все остальные сообщения на ε-переходы в обоих автоматах. Конечные автоматы с данной стадии мы будем именовать "усеченными".

8. Найти, есть ли хоть одна вершина "усеченного" SDL-автомата такая, что, начиная с нее, укладываются **все** трассы **любой длины** с "усеченного" MSC-автомата.

9. При отсутствии противоречий соответствующие вершины должны существовать для всех MSC-диаграмм каждого из объектов. При наличии разногласий это будет выполняться для некоторых MSC-диаграмм некоторых объектов.

Пример. Проверка соответствия SDL-диаграмм MSC-документации при имеющихся отличиях.

Пусть существуют MSC-спецификации 1, 2, 3, 4. Спецификации 1 и 2 изображены на рис. 4, а спецификации 3 и 4 – на рис. 5. Пусть также дана SDL-реализация, приведенная на рис. 6. 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;

[7] О деталях алгоритма проверки можно прочитать в [22].

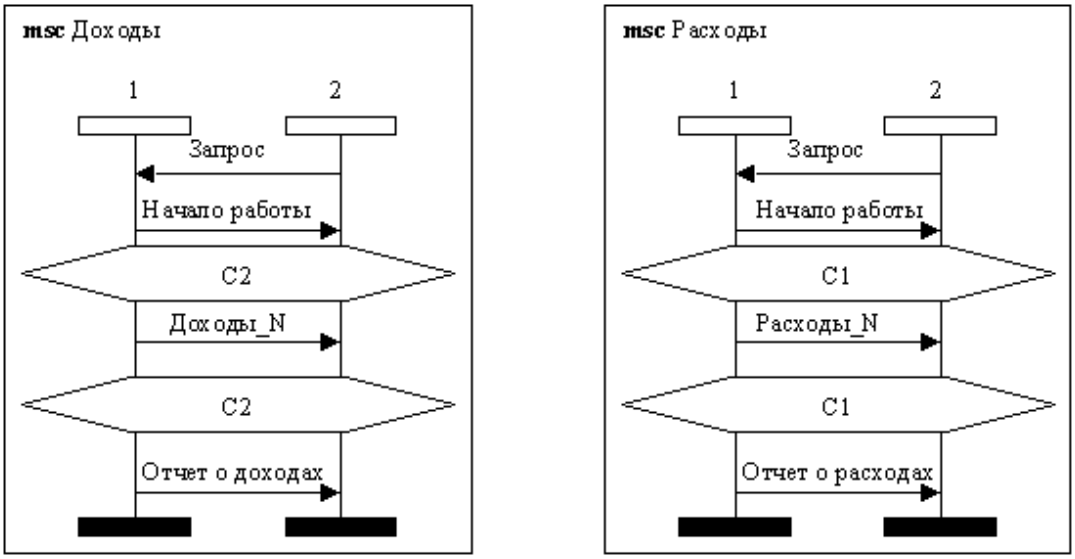

Рис. 8. Детализирующие MSC-диаграммы для примера генерации.

4. удовлетворяет спецификациям 1 и 2 вместе, если начальной является спецификация 2;
5. удовлетворяет спецификации 3;
6. не удовлетворяет спецификации 4.

3.2. Улучшения метода генерации для повышения наглядности SDL-диаграмм

Начнем этот раздел с примера, демонстрирующего различные варианты SDL-кода, который может быть порожден из одного и того же множества MSC-диаграмм. Пусть исходные MSC-диаграммы изображены на рис. 7, 8, а SDL-результаты – на рис. 9, 10. На них для объекта 1 мы строим SDL-диаграмму, реализующую его поведение. Обе построенные SDL-диаграммы с рис. 9 и 10 соответствуют исходным MSC-диаграммам. Достаточно очевидно, какую из двух SDL-диаграммм следует выбрать в качестве реализации – диаграмму с рис. 10.

К сожалению, автоматический алгоритм генерации [8] в качестве результата выдаст другую диаграмму. Причиной этого является то, что в процессе построения накладывается требование, что автомат должен быть детерминированным. Это требование можно ослабить. Достаточно, чтобы он был детерминированным по входящим сигналам – с точки зрения SDL это означает, что входящему сигналу должно соответствовать определенное поведение. Но допустимо, чтобы в начале двух условных веток посылался один и тот же сигнал, что и изображено на рис. 10 после самого верхнего условия.

Обратимся к рис. 11. Это конечный автомат, соответствующий рассмотренному примеру. Видно, что он является более компактным, чем MSC- и SDL-диаграммы. Это происходит потому, что в нем **уже** нет информации об остальных объектах с MSC-диаграммы и **еще** нет информации о логике принятия решений SDL-диаграммы.

Мы предлагаем дать его технологу в качестве промежуточной модели между MSC- и SDL-диаграммами. Благодаря его компактности в него можно внести необходимые изменения – “переразложить” сигналы, некоторые участки слить вместе, некоторые расклеить, указать, что будет выделено в качестве процедур ... А затем проверить его на возможность генерации согласно критерию, сформулированному в начале раздела, и построить SDL-код. Это позволяет интерактивно построить несколько вариантов SDL-моделей, соотвествующих MSC-модели, и выбрать наиболее удобную.

Отличие предложенного решения от имеющихся алгоритмов генерации в том, что мы сумели сохранить возможность построения SDL

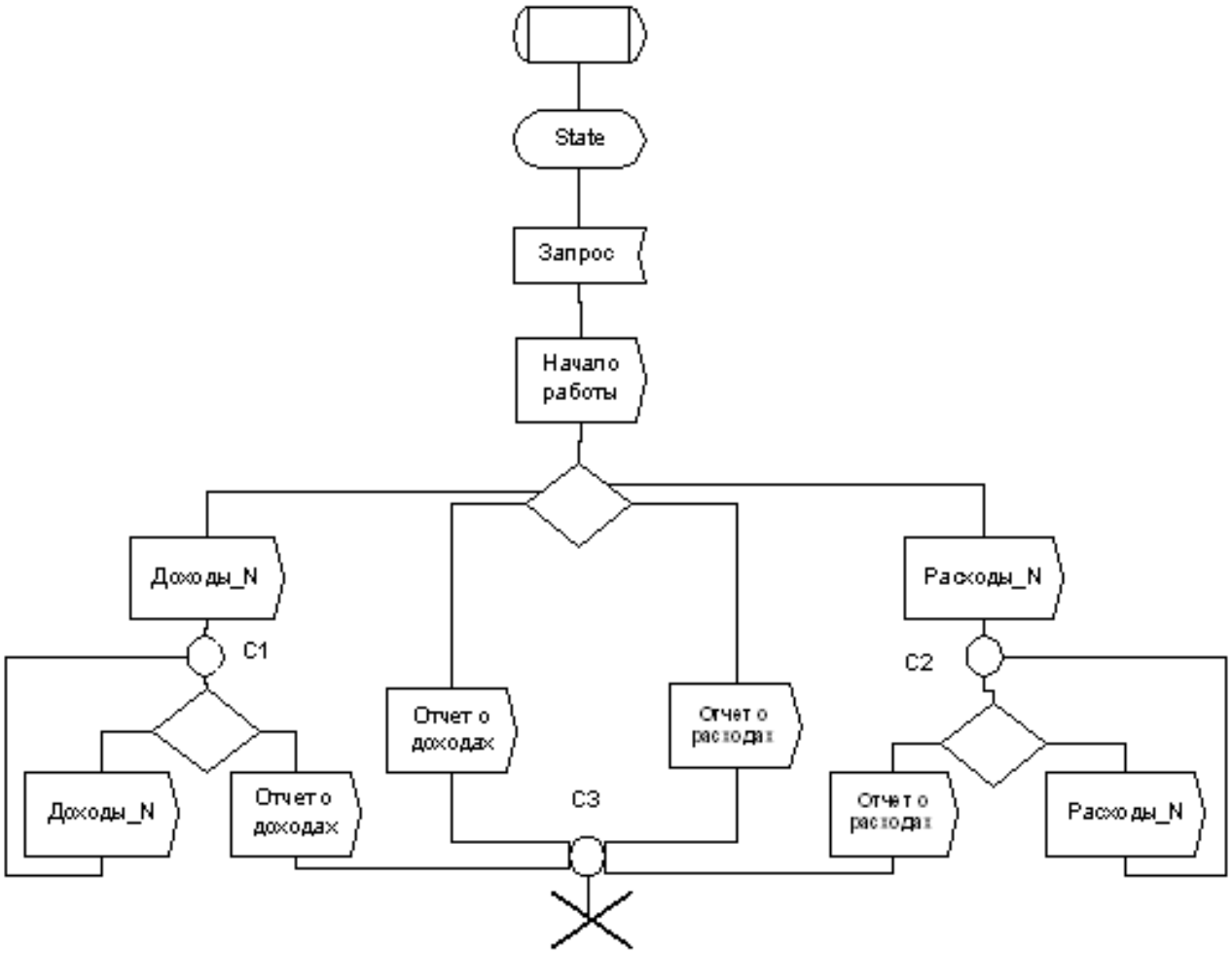

Рис. 9. "Классический" результат генерации.

из всех возможных MSC-диаграмм, но при этом технолог получает не единственную SDL-модель, а выбирает из множества возможных моделей ту, которая лучше соотвествует внутренней логике системы и обладает большей "читаемостью".

3.3. Надстройка над языком MSC для улучшения его описательных свойств

Опыт практического использования MSC-диаграмм при разработке телефонных станций показывает, что очень сильным минусом MSC-диаграмм является их нацеленность только на описание прямых веток. После того, как создана конструкция, описывающая "идеальное" поведение системы, в нее крайне тяжело вставлять ветви, реализующие обработку различных ошибок. Описание небольшого кусочка, реализующего ошибочную ситуацию, влечет за собой добавление еще одного варианта взаимодействия наряду с уже имеющимися на данном уровне. После этого нужно добавить вариант обработки на предыдущем уровне (на том уровне, который детализировался). Зачастую обработка ошибки и на этом уровне не заканчивается, и приходится перерисовывать и еще более вышестоящие уровни. Поэтому описание дополнительного варианта влечет за собой достаточно много "перерисовок", и получающиеся диаграммы быстро растут с перечислением большого количества вариантов поведения. Если проводить параллели с языком программирования Паскаль, то в MSC-диаграммах есть возможность вызвать процедуру, но нет возможности вызвать функцию и получить результат ее выполнения. В

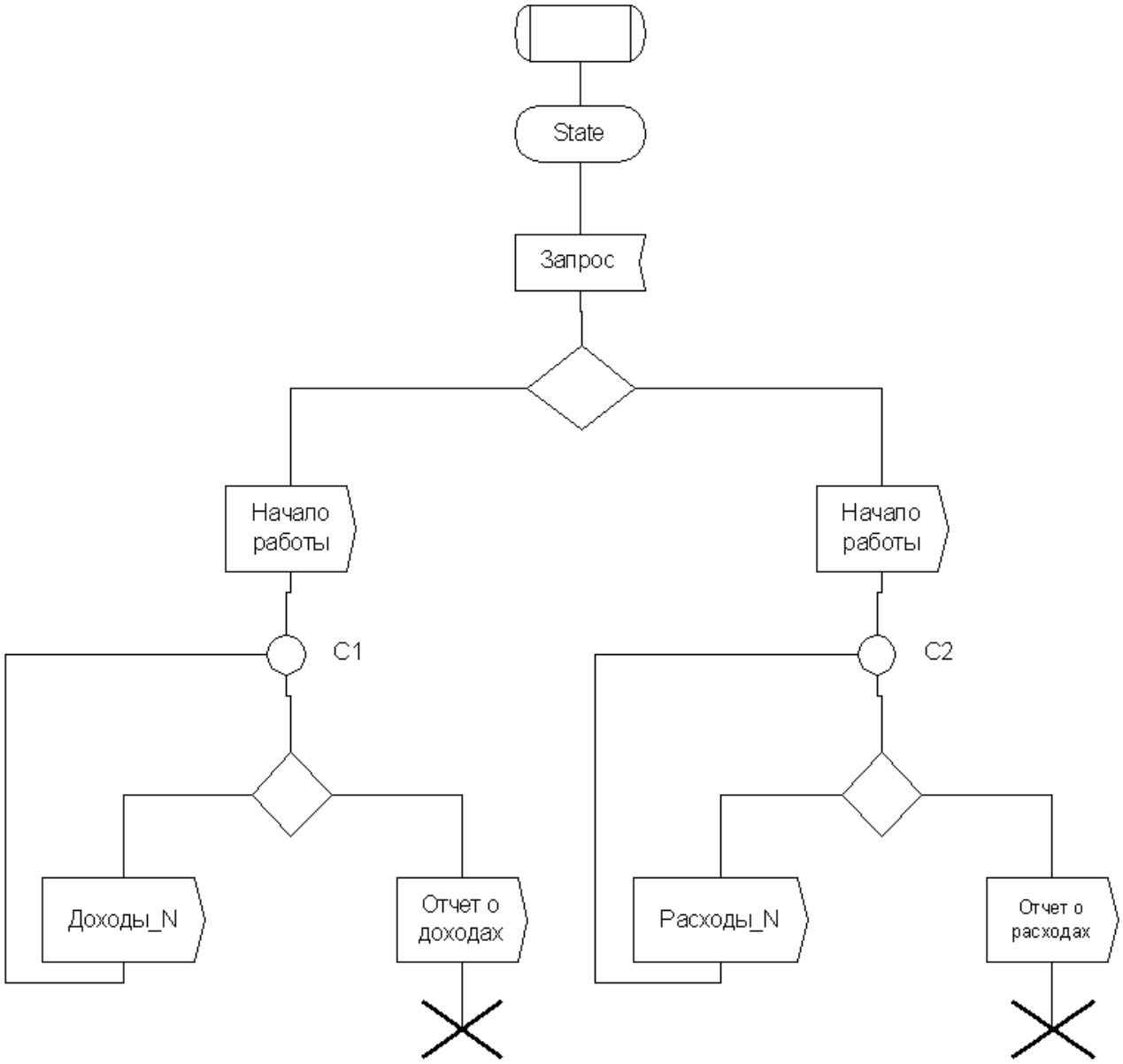

Рис. 10. Наш результат генерации.

реальных же приложениях постоянно приходится предусматривать то, что блок поведения может при исполнении вернуть ряд ошибок. Соответствующими возможностями был расширен язык MSC. Наряду с графическими описаниями предлагается также использовать текстовые описания вида:

```
function сценарий_1(Объект1, Об.2, Об.3) {
   if(сценарий_2(Объект1, Об.2, Об.3)!=1){
      сценарий_3(Об.2, Об.3);
      сценарий_4(Об.2, Об.3);
   } else {
      while(сценарий_5(Объект1, Об.2)==7) {
          if(!сценарий_6(Объект1, Об.2))
                rerurn 0;
      }
   }
   return 1;
}
```

Идея построения заключается в следующем. Пусть у нас есть сценарий, у которого есть 3 варианта выполнения. Для каждого из вариантов построим сценарий, ему соответствующий, и пронумеруем их числами от 1 до 3. Конструк-

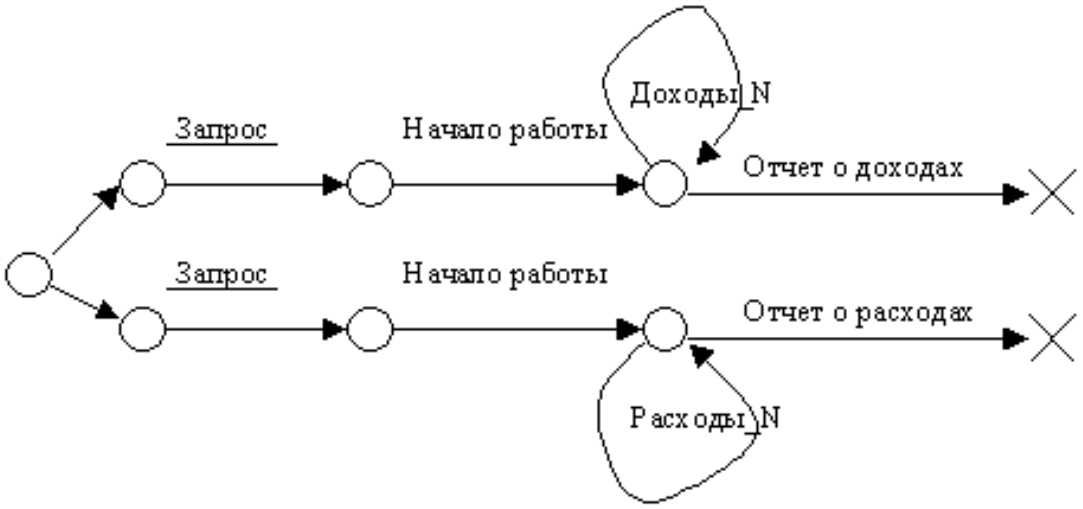

Рис. 11. Автомат, полученный из MSC-диаграмм с рис. 7 и 8 для объекта 1.

цию вида:

```
if(сценарий(...)!=1)
   A
else
   B;
```

мы можем представить в качестве обычной MSC-диаграммы, составленной из сценариев 1, 2, 3, A и B. Аналогичное построение можно составить для ряда других операторов и для сценариев с другими вариантами исполнения. Утверждается, что такой подход лучше соответствует логике протоколов взаимодействия. Для человека обмен сообщениями лучше воспринимается графически, а логика стыковки различных сценариев удобнее описывается с помощью текстовых конструкций.

Описание предложенного расширения

Для текстовых описаний мы вводим операторы:

- **if else**,
- **switch**,
- **for**-определенный – повторить N раз,
- **for**-неопределенный – повторить $0..\infty$ раз,
- **while**,
- **do while**,

с С-подобным синтаксисом. Никакие переменные и числовые операции при этом не вводятся.

Для описания оператора возвращения мы используем слово **return**. В качестве результата мы возвращаем число. Данное число мы анализируем в операторах проверки **if, switch** и **while**. Оно должно быть известно в момент "компиляции", и используется результат выполнения только одного сценария.

В получившемся расширении в качестве "блоков построения" у нас имеются:

- Блоки, ничего не возвращающие:
  - обычные MSC-диаграммы, например, слева на рис. 12;
  - текстовые вставки, в которых нет слова **return**, например, `t_scenario`, описанный ниже. В заглавии таких вставок стоит слово **procedure** с перечислением объектов, участвующих в данном сценарии.

  Их мы будем называть MSC-процедурами. Текстовые вставки можно использовать как аналог текстовых описаний MSC.
- Блоки, которые возвращают результат:
  - MSC-диаграммы дополненные стрелкой возврата (такая диаграмма изображена справа на рис. 12);

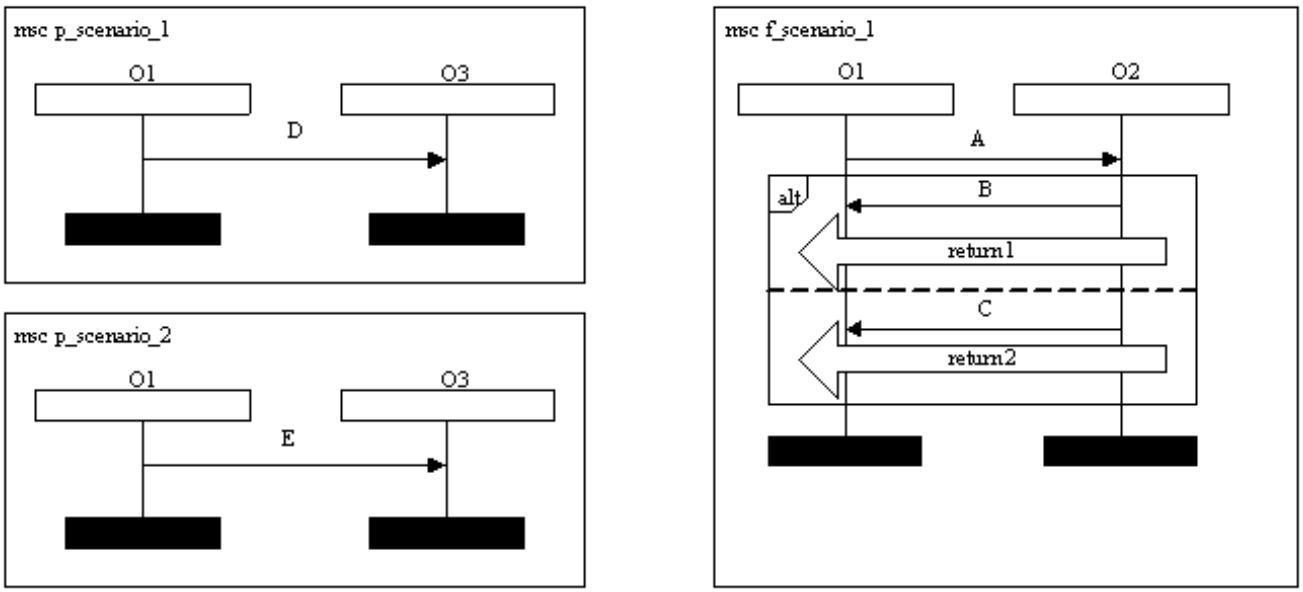

Рис. 12. Две графических процедуры и одна графическая функция.

– текстовые вставки, в которых есть слово **return**, например, вышеупомянутый `сценарий_1`. В заглавии таких вставок стоит слово **function** с перечислением объектов, участвующих в данном сценарии.

Их мы будем называть MSC-функциями. От процедур их отличает то, что к ним добавлен специальный тег, который несет информацию о том, что произошло внутри сценария. Данная информация является неполной и разбивает все возможные варианты поведения на несколько классов.

Приведем пример. Пусть у нас есть две обычных MSC-диаграммы, изображенных на рис. 12, и нижеуказанная текстовая вставка, являющаяся MSC-процедурой.

```
procedure t_scenario(O1, O2, O3) {
  if(f_scenario_1(O1,O2)==1) {
       while( f_scenario_1(O1,O2)>1)
           p_scenario_1(O1,O3);
  } else
       p_scenario_2(O1,O3);
}
```

На рис. 13 приведен аналог текстового описания `t_scenario` в стандарте MSC [2]. Соответствующим образом и следует относиться к предлагаемому расширению – оно дает возможность использовать текстовые расширения с операторами, а соответствует этому некая MSC-диаграмма. Хотя совместимость со стандартами – это важный пункт в настоящее время, но основное применение данного способа описания – это генерация и верификация SDL-моделей, поэтому допустима громоздкость при представлении в качестве стандарта MSC. Разумеется, предполагаемый способ работы – это проектирование высокоуровневых MSC-описаний, не рассматривая, что им соответствует в стандарте MSC. Представление в стандарте нужно лишь для использования решения в существующих средствах.

Сравнение с существующими моделями

Если рассматривать предложенный подход в классе средств описания системы, состоящей из взаимодействующих объектов, то сразу следует исключить ряд графических моделей, таких, как SDL [1], StateChart [5] и текстовые языки (обзор последних можно найти в работе [23]), поскольку они дают взгляд только со стороны одного объекта, а не нескольких. Среди средств UML Sequence [5], UML Collaboration [5], UML Activity [5] и Use Case Maps [7] наш подход вы-

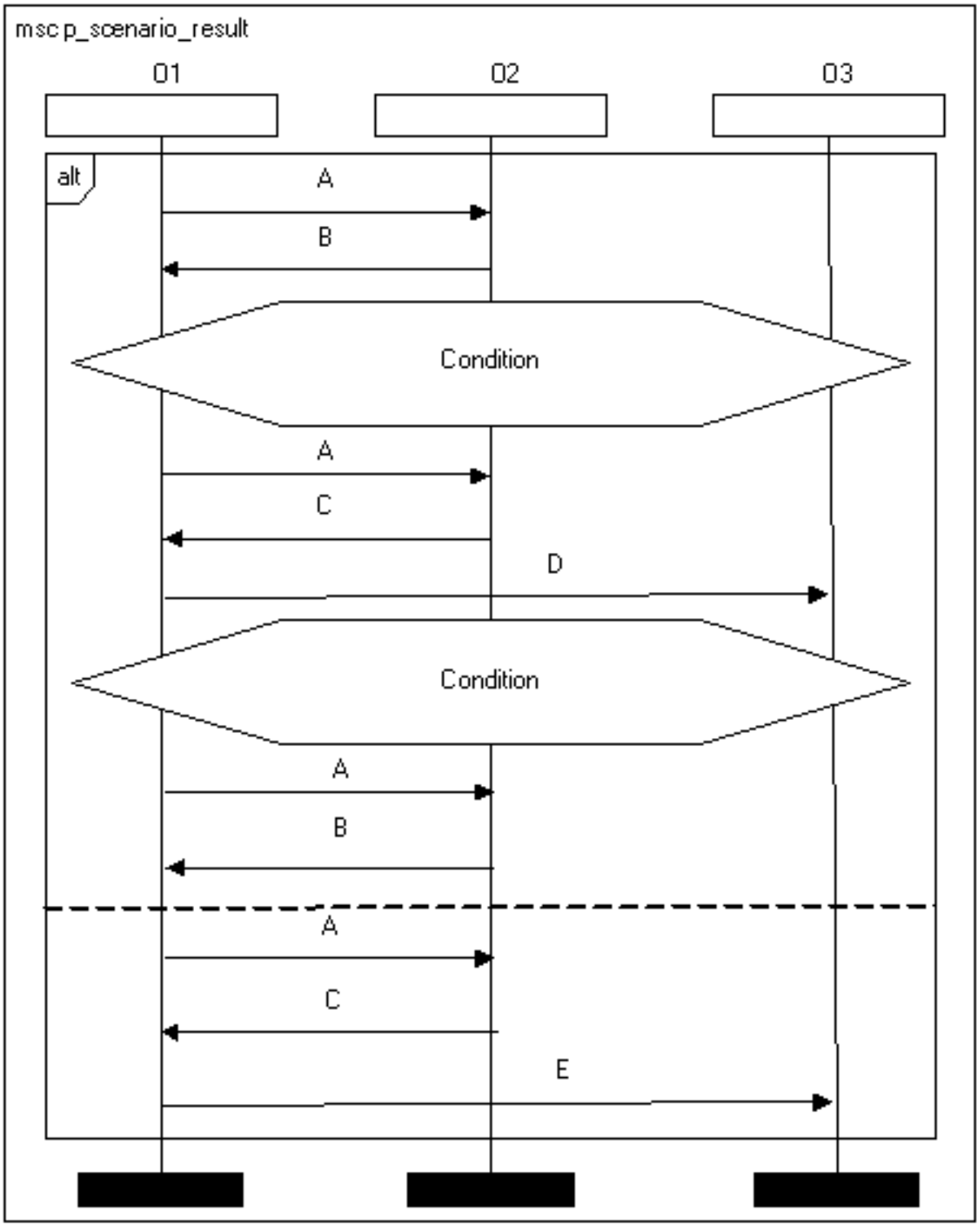

Рис. 13. Результат отображения текстового описания в стандарт MSC.

игрывает за счет большей гибкости и, в результате, за счет возможности получать более полные описания.

Наиболее близкой моделью является другое расширение MSC под названием Life Sequence Charts (LSCs) [24] – попытка приспособить MSC-

сценарии для описания реальных систем. Наш подход предоставляет возможность более удобно описывать различные варианты поведения, поскольку LSCs разрешает выбрать лишь два основных варианта завершения сценария – нормальный и ошибочный. Также, в качестве критики модели LSCs, следует отметить, что это подход, который сильно отходит от стандарта, что требует написания специальных средств для данной модели.

4. ЗАКЛЮЧЕНИЕ

Предложенные подходы по стыковке MSC- и SDL-диаграмм мы объединили в нашей технологии. Графически это изображено на рис. 14.

На нем следует выделить следующие пункты:

1. единый редактор для структур данных. Он гарантирует согласованность статических частей моделей;
2. предложенное расширение MSC-диаграмм. Оно позволяет нам создавать MSC-модели, отражающие жизненные ситуации;
3. функции генерации, которые по MSC-модели позволяют получить скелет SDL-кода и этим существенно сократить время разработки;
4. возможность нарастить SDL фрагментами языка высокого уровня для последующей генерации исполняемого кода;
5. возможности по проверке соответствия SDL-диаграмм MSC-документации при изменении любой из данных моделей.

Вышеописанный подход в рамках нашей технологии позволяет ускорить процесс создания программного обеспечения и повысить качество получаемого продукта за счет максимальной стыковки всех моделей технологии. О стыковке остальных моделей мы писали в [4]. Модели выбраны таким образом, что они достаточно хорошо покрывают область телекоммуникационных систем и систем реального времени. Наш подход решает проблемы, поставленные в разделе 2, и, по мнению авторов, предложенное интегрированное решение позволяет успешно развивать MSC- и SDL-модели с обеспечением их согласованности.

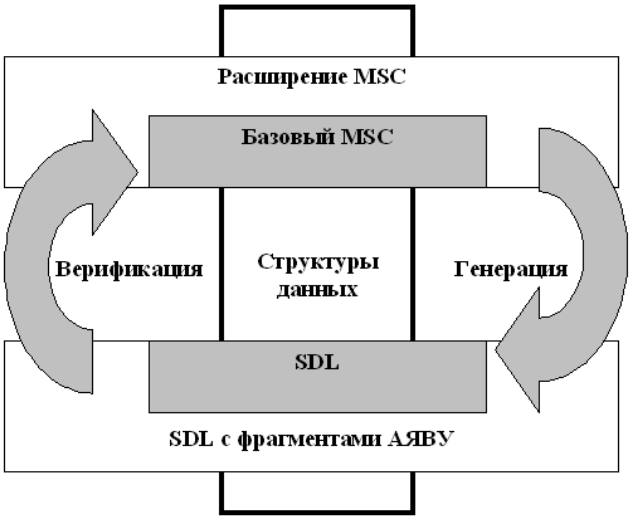

Рис. 14. Объединенный подход к согласованию MSC и SDL в нашей технологии в настоящее время.

Авторы считают, что подобный интегрированный подход также применим в ряде других технологий для заполнения разрыва между MSC-подобной моделью и SDL-моделью. Для этого похожим образом должны быть доработаны следующие моменты:

- при необходимости согласуются статические части моделей;
- модифицируется процедура генерации;
- модифицируется процедура верификации.

Условием на MSC-подобную модель является то, что она должна сводиться к конечно-автоматной. В частности, данный подход применим к другим технологиям, упомянутым в разделе 2.

СПИСОК ЛИТЕРАТУРЫ

1. ITU-T Recommendation Z.100: Specification and description language (SDL). 1999.
2. ITU-T Recommendation Z.120: Message Sequence Chart (MSC). 1999.
3. ITU-T MSC2000R3 Draft Z.120: Message Sequence Charts ITU-T Recommendation Z.120. 1999.

4. *Терехов А.Н.* и др. REAL: методология и CASE-средство разработки информационных систем и ПО систем реального времени // Программирование. 1999. № 5. С. 45–51.

5. *Rumbaugh J., Jacobson I., Booch G.* The Unified Modeling Language Reference Manual. Addison-Wesley, 1999.

6. Сервер группы OMG, занимающейся развитием методологии UML. Стандарты, публикации, средства. http://www.omg.com.

7. Use Case Maps (UCMs) нотация. Определения, публикации, средства. http://www.useCaseMaps.org/index.shtml.

8. *Mansurov N., Zhukov D.* Automatic synthesis of SDL models in Use Case Methodology. Proc. of the 9th SDL Forum. 1999. P. 225–240.

9. *Robert G., Khendek F., Grogono P.* Deriving an SDL specification with a given architecture from a set of MSCs. Proc. of the 8th SDL Forum. 1997. P. 197–212.

10. *Abdalla M., Khendek F., Butler G.* New Results on Deriving SDL Specifications from MSCs. Proc. of the 9th SDL Forum. 1999. P. 55–67.

11. *Li J., Horgan J.* Applying formal description techniques to software architectural design // Computer Communications. 2000. V. 23. № 12. P. 1169–1178.

12. *Khendek F., Vincent D.* Enriching SDL Specifications with MSCs. Proc. of the 2nd Workshop of the SDL Forum Society on SDL and MSC (SAM2000). 2000. P. 305–319.

13. ETR 184. Methods for Testing and Specification (MTS). Overview of validation techniques for European Telecommunication Standards (ETSs) containing SDL. 1995.

14. 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.

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

16. Telelogic Tau 4.2 documentation. March 2001.

17. *Sinclair D., Stone B., Clynch G.* An Object Oriented Methodology from Requirements to Validation. Proc. of the 2nd Object Oriented Information Systems Conference (OOIS'95). Springer, December 1995. P. 265–286.

18. *Holzmann G.* Design and Validation of Computer Protocols. Prentice-Hall, 1991.

19. Описание, публикации, контактная информация средства MOST. http://www.ispras.ru/groups/case/projects.html?2#2.

20. Сайт компании, представляющей средство KLOCWork. http://www.klocwork.com/.

21. Аннотация средства KLOCWork на сайте SDL Forum. http://www.sdl-forum.org/Tools/klocwork.htm.

22. *Соколов В.В.* Проверка соответствия SDL-диаграмм MSC-документации при имеющихся отличиях / Системное программирование. СПб., 2004. С. 366–390.

23. http://mint.cs.man.ac.uk/Projects/UPC/Languages/ConcurrentLanguages.html.

24. *Damm W., Harel D.* LSCs: Breathing Life into Message Sequence Charts. Proc. of the 3rd IFIP International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS'99). Kluwer Academic Publishers, 1999. P. 293–312.