Формальные методы для ускоренной разработки телекоммуникационного программного обеспечения

Н.Н. Мансуров

Зав. отделом Инструментальных средств разработки программного обеспечения
Институт системного программирования РАН

Аннотация

Данная статья обобщает наш опыт разработки инструментальных средств нового поколения, основанных на формальных методах. Целью наших исследований является практическое улучшение процесса разработки программного обеспечения в телекоммуникационном секторе. Мы разрабатываем **методику ускоренной разработки программного обеспечения**, которая охватывает этапы спецификации, проектирования, тестирования и обратной инженерии. Особенность методики заключается в использовании формальных языков спецификации **на ранних этапах** разработки и **автоматическом восстановлении формальных спецификаций** унаследованного телекоммуникационного программного обеспечения. Методика использует наиболее распространенные языки формальных спецификаций, стандартизованные Международным телекоммуникационным союзом (МТС): язык спецификаций и описаний SDL,. язык диаграмм взаимодействия MSC, язык описания тестов TTCN и язык описания данных ASN.1.

В данной статье подробно рассматриваются следующие ключевые части методики:

- спецификация требований к программному обеспечению в виде сценариев с последующей формализацией на языке диаграмм взаимодействия;
- высокопроизводительная валидация требований с использованием моделей на языке SDL;
- автоматический синтез SDL моделей по спецификациям на языке диаграмм взаимодействия;
- детализация высокоуровневых SDL моделей;
- адаптивная генерация программ по SDL моделям;
- автоматическое восстановление SDL моделей по исходным текстам унаследованного программного обеспечения.

1. Введение

В данной статье описываются основные направления научных исследований, выполняемых *отделом инструментальных средств разработки программного обеспечения* ИСП РАН. Отдел был создан в октябре 1997 года на базе двух групп, которые начиная с 1994 года работали в ИСП РАН над созданием различных инструментальных средств для заказчиков из телекоммукационной промышленности [1]. В настоящее время сотрудники отдела разрабатывают методы обратной инженерии [2,5], создают конкретные системы обратной инженерии (совместно с сотрудниками телекоммуникационной промышленности) [7], а также ведут разработку различных инструментальных средств для языков SDL и MSC [3,4,6,8,9,10,11]. Сотрудники отдела принимают участие в преподавании формальных методов в Московском государственном унивеситете [12,13]. Отдел имеет научно-исследовательские связи с группой TSERG (группа исследования методов разработки телекоммуникационного программного обеспечения) факультета информационных технологий Оттавского Университета, а также выполняет работы по контрактам с компаниями Northern Telecom (Канада) и Telelogic AB (Швеция).

Главным направлением научно-исследовательской деятельности отдела является:

- разработка **методики ускоренной разработки телекоммуникационного программного обеспечения**, состоящей из набора методов спецификации,

проектирования, тестирования и обратной инженерии телекоммуникационного программного обеспечения, причем особенностью методики является использование языков формальной спецификации на ранних этапах разработки и автоматическое восстановление формальных спецификаций унаследованного программного обеспечения, а также

- создание **открытой архитектуры**, задающей набор формальных языков и инструментальных средств, поддерживающих методику ускоренной разработки и позволяющих масимально ее автоматизировать, делая возможным более широкое внедрение формальных методов в индустриальную практику.

Актуальность нашей исследовательской программы определяется следующими обстоятельствами. Для современной телекоммуникационной промышленности характерны частое обновление технологий, а также сильная конкуренция. В этих условиях решающим фактором коммерческого успеха становится *время выхода на рынок* [13]. Дополнительные требования более традиционны. Они включают в себя более высокое качество продуктов, лучшее соотношение цена.производительность и более низкие производственные затраты [13,21]. По устоявшемуся мнению, использование формальных методов и соответствующей инструментальной поддержки является важным условием достижения этих целей.

Международный телекоммуникационный союз разработал и стандартизовал семейство формальных языков спецификации. К ним относятся язык спецификаций и описаний SDL [14], язык диаграмм взаимодействия MSC [15], язык спецификации тестов TTCN [16], а также язык спецификации данных ASN.1 [17]. Язык SDL является наиболее успешным стандартизованным формальным языком, используемым в телекоммуникационной промышленности. В настоящее время доступны коммерческие инстументальные средства, позволяющие анализировать SDL спецификации, проводить их валидацию используя алгоритмы анализа состояний, автоматически генерировать тесты на языке TTCN по SDL спецификациям, а также автоматически генерировать исполняемые программы для операционных систем реального времени [18]. Ряд экспериментальных исследований продемонстрировал, что использование CASE-систем на базе языка SDL в промышленных проектах позволяет повышать качество программного обеспечения, снижает стоимость разработки, и, самое главное,позволяет снижать сроки разработки на 20-30%. Наиболее существенные успехи использования языка SDL в телекоммуникационной индустрии связаны с этапами системного проектирования, детального проектирования и автоматической кодогенерации [3,4,6,9,10,11,29], а также с формальной верификацией и тестированием программного обеспечения [19,20].

Вместе с тем, существует ряд барьеров для более широкого внедрения формальных методов в промышленности. По нашему мнению, наиболее существенными из них являются поддержка ранних этапов разработки и поддержка интеграции с унаследованным программным обеспечением. Основной задачей нашего исследования мы видим устранение указанных барьеров.

Существует значительный разрыв между математически-ориентированными формальными методами и практикой ранних стадий разработки программного обеспечения [21]. Обычно проектирование новой системы начинается с этапа описания требований, на котором, как правило, делаются серии предварительных неформальных набросков, имеющих целью нащупать правильное решение. Наброски часто переделываются и лишь немногие из них переносятся в проектную документацию. Написание проектной документации становится возможным только после того, как сформировалась концепция будущей системы. На ранних этапах разработки методы формальной спецификации и валидации оказываются практически бесполезными для разработчиков, поскольку требуемый уровень формальности и точности еще не достигнут. В связи с этим, на ранних этапах разработки оказывается возможным формализацизовать только очень абстрактные свойства системы. Цена такой ранней формализации оказывается достаточно высокой, а результаты – слишком незначительные [21].

Напротив, широкое распространение проибретают так называемые сценарные

методы проектирования [22,30]. Особенностью сценарных методов является описание требований в виде сценариев использования. Для формализации сценариев могут использоваться диаграммы взаимодействия (MSC) или так называемые диаграммы действий языка UML [30]. Язык диаграмм взаимодействия является особенно привлекательным, поскольку он широко распространен в телекоммуникационном секторе и имеет стандартизованную формальную семантику. Однако в современных коммерческих CASE системах моделирование с использованием языка диаграмм взаимодействий поддерживается недостаточно.

Мы считаем, что инструментальная поддержка формальных языков, пригодных для ранних этапов разработки должна привести к значительному ускорению процесса разработки. Основной особенностью разрабатываемой нами методики ускоренной разработки программного обеспечения является использование автоматического синтеза исполняемых SDL спецификаций по MSC спецификациям. Дополнительно, наша методика поддерживает расширения языка диаграмм взаимодействия, предназначенные для описания работы с данными. Синтез осуществляется инструментальным средством, которое мы назвали Московский Синтезатор (MOST-SDL) [8]. Синтезированные SDL спецификации могут быть использованы для валидации требований, а затем постепенно детализированы до уровня исполняемых моделей. Прочие шаги нашей методики включают в себя быструю генерацию тестов по SDL спецификациям и автоматическую генерацию кода по детализированным SDL спецификациям [13].

Помимо поддержки ранних этапов разработки, существует еще одина важная проблема, которая должна быть решена для внедрения CASE-системы, основанных на формальных методах, в промышленную практику. Формальные методы применимы только для таких проектов, в которых программное обеспечение разрабатывается «с нуля». Однако, большинство реальных промышленных проектов используют уже написанное, так называемое унаследованное (базовое) программное обеспечение. Базовое программное обеспечение нуждается в сопровождении, расширении и адаптации к изменениям в окружении. Новые проекты, как правило, основаны на значительном повторном использовании базового программного обеспечения. Для внедрения формальных методов в промышленности необходимо разработать эффективные методы интеграции компонентов, полученных в среде CASE систем нового поколения и более старых базовых унаследованных компонентов. Унаследованное программное обеспечение обычно разрабатывалось по устаревшим методам. Такие системы часто состоят из смеси высокоуровневого и системного кода, смеси различных языков программирования, архитектур и стилей. Одной из наиболее серьезных проблем, возникающих при использовании унаследованного кода, является отсутствие документации, описывающих проектные решения. До недавнего времени, требование интеграции с унаследованным программным обеспечением являлось барьером для широкого использования формальных методов в промышленности [5].

Для преодоления «барьера унаследованного кода» требуется разработать автоматизированные методы обратной инженерии, которые помогут значительно сократить затраты на восстановление формальных спецификаций базового программного обеспечения. По нашему мнению, эффективные методы восстановления SDL спецификаций унаследованного программного обеспечения сделают возможным достижение следующих целей:

- более глубокое понимание работы унаследованных компонентов путем исполнения SDL модели, что дает более интуитивное представление результатов и не требует использования дорогостоящего спецоборудования;
- автоматизированная генерация тестов для унаследованного программного обеспечения;
- анализ и валидация формальных спецификаций расширений базового программного обеспечения;
- анализ взаимодействия расширений;
- автоматизированная генерация тестов для расширений;
- автоматическая генерация кода для расширений. Автоматически генерируемый код может быть

перенацелен на различные целевые языки программирования (например, C, C++,CHILL), а также на работу с различными системами реального времени (например, pSOS, VxWorks и т.д.).

В данной статье мы представляем нашу методику так называемого динамического сценарного восстановления спецификаций унаследованного телекоммуникационного программного обеспечения в виде SDL моделей.

Наш подход к восстановлению SDL спецификаций включает в себя следующее:

- инструментирование унаследованного кода при помощи семантических датчиков, расставляемых в ключевых местах кода, выявляемых на основе статического анализа архитектуры системы;
- выбор представительных сценариев из базового набора тестов и других источников;
- выполнение сценариев на инструментированной системе и сбор трасс, с последующим переводом трасс на язык диаграмм взаимодействия с состояниями;
- синтез SDL моделей по набору диаграмм взаимодействия при помощи московского синтезатора MOST-SDL [5].

Статья имеет следующую организацию. Раздел 2 представляет обзор нашей методики ускоренной разработки программного обеспечения. Разделы 3-6 посвящены вопросам прямой инженерии и описывают применение методики на этапах описания требований, проектирования и тестирования телекоммуникационного программного обеспечения. Раздел 3 описывает вопросы ранней формализации требований, а также методику высокопроизводительной валидации требований с использованием SDL моделей. Раздел 4 описывает центральный компонент нашей методики – автоматический синтез SDL спецификаций по сценариям. Раздел 5 посвящен вопросам детализации синтезированных SDL моделей. Раздел 4 обобщает наш опыт по адаптивной кодогенерации по SDL спецификациям. Раздел 7 описывает наш подход к восстановлению SDL спецификаций унаследованного телекоммуникационного программного обеспечения. В разделе 8 формулируются выводы описываются некоторые направления дальнейших исследований.

2. Ускоренная разработка телекоммуникационного программного обеспечения

В данной статье описывается современная прагматическая методика разработки программного обеспечения, основанная на широко распространенных стандартизованных формальных языках спецификации MSC, SDL и TTCN. Целью нашего исследования является демонстрация того, что современные формальные методы и поддерживающие их CASE-системы могут существенно сократить сроки разработки промышленного телекоммуникационного программного обеспечения и, тем самым, сократить время выхода продуктов на рынок, что является важнейшим определяющим фактором успеха в телекоммуникационном секторе.

До недавнего времени, инструментальные системы поддержки разработки (CASE-системы) не находили широкого применения в промышленных организациях. Однако современные CASE-системы оказываются гораздо более пригодными для работы с крупными промышленными проектами. В данной статье мы описываем методику применения CASE-систем для сокращения времени разработки программного обеспечения.

Рис. 1. Перераспределение времени этапов

Многие современные методики разработки программного обеспечения используют принцип удлинения начальных этапов для сокращения общего времени разработки и времени выхода на рынок. На рис. 1 представлено перераспределение времени различных этапов, происходящее в результате применения CASE-систем, поддерживающих формальные методы [13]. Мы утверждает, что увеличение времени начальных этапов до 17%, связанное с построением, валидацией и сопровождением формальной спецификации на SDL и генерации тестов по формальной спецификации может приводить к

сокращению общего времени разработки на 30-50% [13].

Известно, что использование формальных методов может приводить к значительному улучшению качества программного обеспечения, что до недавнего времени считалось их главным премуществом. Поэтому основными заказчиками формальных методов выступали секторы промышленности с повышенными требованиями к безопасности программного обеспечения. Однако в данной статье исследуются возможности формальных методов по сокращению времени разработки программного обеспечения, ускорение выхода продукта на рынок. Улучшение качества программ также оказывает прямое влияние на сокращение времени разработки за счет снижения затрат на исправление ошибок и упрощение сопровождения.

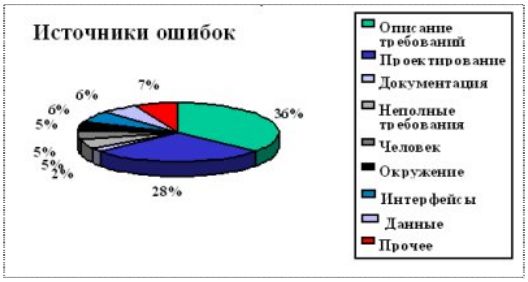

Рис. 2. Источники ошибок

Значительные сокращения времени разработки могут быть достигнуты за счет снижения затрат на поиск и исправление ошибок в программном обеспечении. По материалам недавних исследований, проведенных университетом Западной Вирджинии и ВВС США [13], источником большинства ошибок являются ранние этапы разработки (см. рис. 2). Причиной 36% ошибок является неправильное истолкование требований, 5% ошибок происходит из-за неполноты требований, 28% ошибок делаются на этапе проектирования. На остальные источники ошибок (взятых вместе) приходится остальные только 31%.

С другой стороны, большинство ошибок были обнаружены (и исправлены) на поздних этапах процесса разработки (см. Рис. 3). Так 13,83% ошибок были обнаружены на этапе полетных испытаний, 51,06% ошибок были обнаружены на этапе сборки изделия, 15,96% - на этапе сборки программного обеспечения. Только 9,57% ошибок были обнаружены на этапе анализа требований.

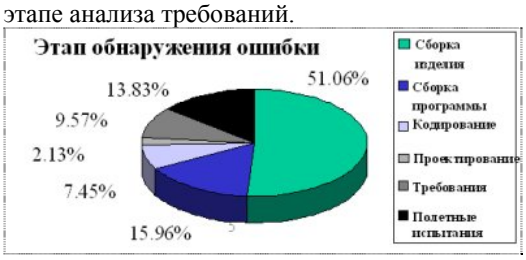

Рис. 3. Этап обнаружения ошибки

Особенностью нашего исследования является преодоление препятствий для внедрения формальных методов на ранних этапах разработки (сбор и описание требований), а также на поздних этапах (сопровождение). На средних этапах разработки наша методика предполагает использование традиционных преимуществ формальных методов. Эти процессы представлены на рис. 4.

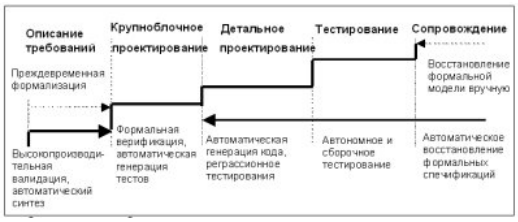

Рис. 4. Процессы разработки

Рис. 5. Этапы методики и связи

Более подробный обзор нашей методики представлен на рис.5. На этом рисунке показаны этапы методики (справа), ключевые модели (в прямоугольниках), а также наиболее важные обратные связи, обеспечиваемые инструментальными средствами (пунктиры). Зависимости между моделями показаны стрелками. На рисунке опущены зависимости и обратные связи между моделями одного и того же этапа, а

также обратные связи между моделями соседних этапов. Мы различаем две различные точки зрения моделирования [22,30,13]: функциональная (слева) и структурная (справа). Структурная точка зрения обычно описывает то, из каких компонентов состоит программа и какие связи имеются между этими компонентами. Функциональная точка зрения описывает то, каким образом компоненты взаимодействуют для обеспечения требуемого поведения.

Информационные обратные связи обеспечивают возможность итеративной разработки. По нашему мнению, ускорение процесса разработки должно быть основано на использовании таких итераций. Ключевой особенностью нашего подхода является как можно более раннее использование формальных методов для того, чтобы иметь возможность поддерживать наиболее важные обратные связи и вести на их основе итеративную разработку.

Заметим, что в типичном промышленном процессе разработки формализация в лучшем случае начинается с этапа детального проектирования. В случае использования языка SDL системная модель представляется набором SDL блоков, а детальная модель – набором SDL процессов. Переходы к последующим этапам осуществляютсяя вручную, за исключением последнего (перехода от детальной модели к реалиизации). Данный переход частично осуществляется с помощью автоматической кодогенерации. В такой ситуации единственной обратной связью, которую может обеспечить CASE-система, является связь между реализацией и детальной моделью. Остальные обратные связи оказываются практически бесполезными, т.к. соответствующие модели не формализованы, а ручная модификация детальной модели слишком дорогостоящая.

Наша методика ускоренной разработки программного обеспечения представляет собой итеративный объектно-ориентированный процесс, включающий в себя следующие шаги:

- описание требований в виде сценариев, формализованных на языкке диаграмм взаимодействия;
- высокопроизводительная валидация требований с использованием исполнимых моделей на языке SDL;
- синтез моделей требований на языке SDL по сценариям;
- постепенная детализация моделей требований;
- адаптивная кодогенерация по SDL спецификациям;
- высокопроизводительное проектирование тестов ня языке TTCN;
- автоматизированное восстановление SDL моделей по исходным текстам унаследованной системы.

Рис. 6. Отдельные шаги методики

Отдельные шаги методики ускоренной разработки представлены на Рис. 6. Как показано на рисунке, все шаги поддерживаются инструментальными системами для языов SDL, MSC, TTCN.

3. Описание и валидация требований

3.1. Сценарии использования

В настоящее время широко распространены методики разработки программ, основанные на концепции сценария использования. Основными понятиями концепции сценария использования являются понятия актора и сценария [22]. Актор представляет собой абстрактную сущность, являющуюся внешней по отношению к разрабатываемой системе. По определению, актор взаимодействует с проектируемой системой для достижения определенных целей. Сценарий представляет собой типичную (иллюстративную) последовательность взаимодействий между одним или несколькими акторами и системой, в результате чего один из акторов достигает одной из своих целей. Такая цель называется целью сценария. Один сценарий может описывать нексолько альтернативных последовательностей событий. За последнее время был разработан целый ряд методик, основанных на концепции сценария

[22,25,30,12].

Все методики, основанные на концепции сценария использования, описывают требования к проектируемой системе в виде функциональных сценариев, однако используют различные способы представления сценариев. Согласно [22], отношения между сценариями и акторами представляются на неформальной диаграмме. Сценарии задаются в виде таблиц [12], текста на естественном языке [22], диаграмм действий языка UML [30]. В работе [25] предлагается формализация сценариев на языке диаграмм взаимодействия.

3.2. Высокопроизводительная валидация требований

В данном подразделе рассматривается методика валидации требований к программному обеспечению. Валидацией требований называется процесс выявления ошибок в требованиях к проектируемой системе, формулируемых заказчиками. Как было отмечено в разделе 2, обратная связь от реализации к требованиям является решеющей в процессе разработки программного обеспечения. В соответствии с предлагаемой методикой ускоренной разработки (см. рис. 7), валидация требований представляет собой итеративный процесс, состоящий из следующих шагов:

1. Провести формализацию требований в виде сценариев.
2. Создать исполняемую модель требований
3. Создать валидационные сценарии
4. Прогнать валидационные сценарии через модель требований
5. Оценить результирующее поведение модели. При этом возможны два варианта:
   - 5.1. Поведение является допустимым. В этом случае валидационный сценарий может быть добавлен к множеству исходных сценариев.
   - 5.2. Поведение не является допустимым. В этом случае валидационный сценарий отвергается. Эта ситуация означает наличие ошибки или противоречия в требованиях, либо пропущенное требование. Отвергнутый сценарий должен быть преобразован в сценарий использования и добавлен к исходному набору сценариев использования (возможно, после удаления противоречий и т.д.).
6. Проверить условия завершения и, если необходимо, начать новую итерацию с шага 2.

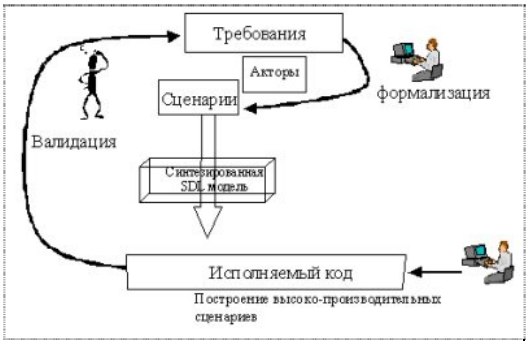

Рис. 7. Валидация требований

Введем необходимую терминологию для обсуждения *валидационных сценариев*. Будем различать *первичные сценарии* (штатное поведение) и *вторичные сценарии* (нештатное поведение, исключительные ситуации, коллизии). Все функциональные сценарии являются, по-определению, первичными. Вообще, большинство сценариев, описывающих требования, являются первичными.

Под производительностью сценария будем понимать количество ошибок, обнаруженное (или ожидаемое) при выполнении данного сценария. Первичные сценарии обладают низкой производительностью, поскольку они описывают достаточно хорошо изученные ситуации. Напротив, вторичные сценарии являются средне- или высокопроизводительными, поскольку они описывают менее хорошо документированное поведение. Производительность вторичных сценариев оказывается более высокой, поскольку решения проектировщиков в ситуациях, описываемых такими сценариями, могут существенно отличаться от желаний заказчиков либо поведение может оказаться недетерминированным. Наша методика предполагает использование высокопроизводительных сценариев, а также осуществление регулярного контроля покрытия, что позволяет оценивать качество валидации. Высокопроизводительная валидация происходит под управлением риска, когда усилия по валидации сосредотачиваются на компонентах с наибольшим риском.

Понятие *риска* связано с понятием *ожидаемой стоимости ошибки*. По определению, риск R вычисляется как PF×CF

(где PF есть вероятность ошибки; CF стоимость ошибки). *Производительность сценария* пропорциональна числу ошибок, обнаруженных в результате валидации данного сценариея. С другой стороны, производительность сценария может быть соотнесена с количеством риска, снятого в результате рассмотрения данного сценария.

Низкопроизводительный сценарий имеет низкую вероятность обнаружения ошибки, поскольку такие сценарии, как правило, хорошо понятны как заказчикам, так и разработчикам. Напротив, высокопроизводительные сценарии имеют высокую вероятность обнаружения ошибки, в силу того, что они соответствуют вторичным сценариям, т.е. нештатному поведению, обработке аварийных ситуаций и т.п. Обычно, такие сценарии оказываются гораздо менее хорошо изучены разработчиками.

Рис. 8. Классификация сценариев

В [13] нами была предложена следующая классификация сценариев по отношению к их производительности (см. рис. 8).

Высокопроизводительная валидация может рассматриваться к валидация, управляемая моделью ошибок. Рассмотрение типичных ошибок и типичных областей, связанных с высоким риском, может существенно повышать производительность процесса валидации. Для построения высокопроизводительных сценариев можно построить модель ошибок (на основе некоторой гипотезы об источниках ошибок), а затем выводить сценарии непосредственно на основе такой модели. В случае достаточно полной модели ошибок, критерием покрытия может служить покрытие самой модели ошибок.

Важным источником повышения производительности сценариев является валидация поведения на граничных условиях. Общая стратегия построения высокопроизводительных сценариев заключается в рассмотрении представителей допустимых и недопустимых классов эквивалентности для входных и выходных величин. Одна часть сценариев должна обеспечить общее покрытие допустимых классов эквивалентности, в то время как другая часть должна включать в себя представителей недопустимых классов эквивалентности непосредвтсвенно на границе и вблизи нее.

Высокопроизводительная валидация требований к программному обеспечению является разновидностью тестирования, адаптированного для ранних этапов разработки. Большинство рекоммендации относительно тестирования программ применимы применимы и к задаче валидации требований. Валидация не может гарантировать полное отсутствие ошибок. Поскольку исчерпывающее тестирование невозможно по стоимостным соображениям, нужно стремиться максимизировать производительность тестов, т.е. максимизировать количество ошибок обнаруженных набором тестовых сценариев заданного объема.

3.3. Автоматический синтез исполнимых моделей требований на SDL

Основу предлагаемой методики ускоренной разработки программ составляет шаг автоматического синтеза исполняемой SDL модели на этапе анализа требований (Рис. 9). Входной информацией для синтеза является сценарная модель, формализованная на языке диаграмм взаимодействия. Единственная структурная информация, присутствующая на данном этапе, состоит из описания множества акторов (представленных отдельными экземплярами на MSC диаграммах). Описание поведения представлено в виде множества частичных функциональных сценариев, описывающих наиболее типичные взаимодействия между внешними акторами и системой. Важная дополнительная информация может быть задана в виде описания потоков данных, передаваемых по сценариям. Для задания этой информации нами были введены специальные расширения языка диаграмм взаимодействия (см. подраздел 4.1.). Так

называемая синтезированная модель требований представляет собой одну из возможных проекций информации, формализованной на этапе сбора требований, на этапы детального проектирования и реализации. Это позволяет задействовать все обратные связи, показанные на рис. 9, поскольку информация становится доступной непосредственно по мере составления требований, что позволяет осуществлять итеративную их разработку.

Автоматический синтез исполняемы моделей требований на языке SDL по MSC спецификациям обладает следующими преимуществами:

- путем исполнения валидационных сценариев на SDL модели становится возможной высокопроизводительная валидация требований;
- MSC спецификации могут разрабатываться независимыми группами специалистов, поскольку архитектурная целостность системы будет автоматически обеспечиваться при синтезе;
- автоматический синтез снимает необходимость в регрессионном тестировании, поскольку сценарии, признанные допустимыми, добавляются к набору валидационных сценарием и используются на следующих итерациях как входная информация для синтеза;
- синтезатор позволяет производить раннее обнаружение определенных классов ошибок;
- синтез позволяет исследовать различные правила композиции сценариев (последовательная композиция, параллельная композиция и т.д.);
- MSC модели обладают высокой модельностью, поэтому они гораздо более предпочтительны в сопровождении, обладают высоким потенциалом для повторного использования.

Синтезированная модель требований может быть использована для генерации дополнительных сценариев, которые оказываются длиннее исходных валидационных сценариев и, тем самым, имеют большую ценность для валидации [27]. Исполнение синтезированной модели требований позволяет быстро обнаруживать неполноту исходных требований и противоречия в требованиях. Это происходит благодаря тому, что дополнительные сгенерированные валидационные сценарии осуществляют значительную вариацию поведения, включая нештатные ситуации. Такие сценарии оказываются более производительными, по сравнению с исходными.

Синтезированная модель требований на языке SDL может быть использована для автоматической генерации тестов на языке TTCN [19,20].

4. Синтез SDL спецификаций по сценариям

В данном разделе будут описаны детали автоматического синтеза SDL моделей по сценариям, формализованным на языке диаграмм взаимодействия. Будут рассмотрены вопросы формализации сценариев, расширения языка диаграмм взаимодействия для описания обработки данных, ключевые понятия, используемые алгоритмом синтеза. Описание алгоритма синтеза будет проиллюстрировано небольшим примером.

Особенности нашего подхода включают в себя следующее:

- формализация сценариев использования при помощи диаграмм взаимодействия высокого уровня (HMSC);
- генерация конечных автоматов по сценариям;
- спецификация потоков данных внутри сценариев с использований специальных расширений языка диаграмм взаимодействия;
- автоматический синтез моделей на языке SDL.

4.1. Формализация сценариев на языке диаграмм взаимодействия

Наш подход к формализации сценариев основывается на следующих принципах:

- каждый сценарий формализуется на так называемой базовой диаграмме взаимодействия (bMSC) [15];
- связи по управлению между сценариями формализуются припомощи диаграмм взаимодействия ввысокого уровня *High-Level MSCs* (HMSC) [15]. Связи по управлению между сценариями включают в себя альтернативные части сценариев, повторяющиеся части сценариев, а также отношение использования и отношение расширения одного сценария другим [22];

- связи по данным между сценариями формализуются при помощи наших расширений языка даиграмм взаимодействия [8];
- используется определенное правило композиции сценариев [8]:
  - Последовательная композиция сценариев (внешний актор выбирает некоторый сценарий, который затем выполняется до полного завершения)
  - Параллельная композиция (одновременно могут выполняться несколько различных сценариев)
  - Множественная композиция (одновремено выполняются несколько экземпляров одного и того же сценария для разных акторов)

Для формализации сценраиев используется расширенный язык диаграмм взаимодействия [15]. Каждый сценарий формализуется диаграммой взаимодействия высокого уровня, описывающей связи по управлению между сценариями. Специальные расширения языка диаграмм взаимодействия описывают потоки данных внутри сценариев (так называемые локальные потоки данных), а также потоки данных между различными сценариями (так называемые глобальные потоки данных). Рассмотрим наши расширения более подробно [8].

1. **Определения переменных**. Допускаются определения переменных различных типов. Семантика звимствована из языка SDL. Графически, определение переменной задается в текстовом символе на любой диаграмме. По построенияю, локальная копия этой переменной будет присутствовать в каждом процессе. При определении переменных используется следующий упрощенный синтаксис языка SDL:
   `<variable definition> ::= dcl <var_name> <type>;`
2. **Действия.** В символах действия диаграмм мы допускаем использование операций над локальными переменными. Семантика операций наж переменными заимствуется из языка SDL. При определении переменных используется следующий упрощенный синтаксис языка SDL:
   `<assignment> ::= <var_name> := <expr>`
   `<function call> ::= <func> (<expr1>,…,<exprn>)`
3. **Параметры сообщений**. Мы допускаем использование параметров соощений. Синтаксически, в качестве параметров сообщений допускаются только имена переменных. Семантика параметров сообщений заимствуется из языка SDL.
4. **Параметры создания процессов**. Акторы могут передавать параметры при создании экземпляров процессов. В качестве параметров при создании процессов могут использоваться только имена переменных. Семантика параметров сообщений заимствована из языка SDL.
5. **Локальные состояния**. Существенным расширением языка диаграмм взаимодействия является использование локальных состояний, связанных с проверкой значений локальных переменных. Графически, локальные проверки задаются локальным состоянием, к которому присоединен дополнительный комментарий, содержащий булевское выражение.. Семантика локального состояния таковы, что все последующие события выполняются только тогда, когда условие, указанное при состоянии, является истинным. Синтаксически, выражения, участвующие в локальных состояниях, могут иметь следующий вид:
   `<boolean expression> ::= <var_name> <op> <var_name> | <var_name> <op> <const>`
   Альтернативные последовательности событий могут быть заданы на другой диаграмме после локального состояния с тем же именем, но другим выражением. Все выражения в локальном состоянии с одним и тем же именем, должны быть взаимно-исключающими.
6. **Таймеры**. Последовательность событий *set* и *timeout* на диаграммы взаимодействия может быть использована для задержки во времени выполнения сценария. При нештатном выполнении сценария такая задержка может приводить к ошибке. Таймеры с параметрами не поддерживаются.

Понятие потоков данных через сценарии проиллюстрировано на рис. 10 и рис. 11. На рис. 10 приведены два локальных потока данных через сценарий abc are shown (пунктирные линии). Первый поток включает в себя следующие события: a: in x(p,q) from env; a: create b(p,q); b: out w(r) to env;

Заметим, что объекты B и C используют различные (локальные) копии переменной r. Таким образом, параметр сообщения, посылаемый объектом B в не обязательно равен p+1.

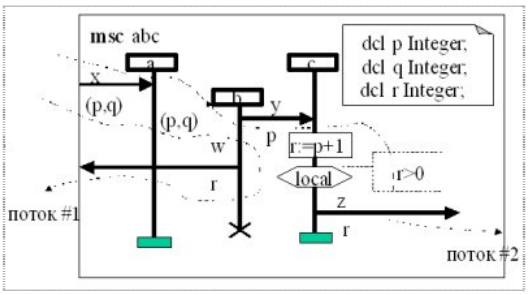

Рис. 10. Потоки данных через сценарий (1)

Второй поток данных включает в себя следующие события: a: **in** x(p,q) **from env**; a: **create** b(p,q); b: **out** y(p) **to** c; c: **in** y(p) **from** b; c: **action** ‘r:=p+1’; c: **out** z(r) **to env**.

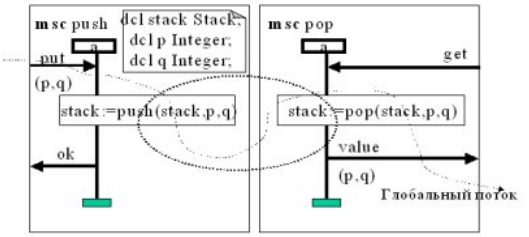

Рис. 11. Потоки данных через сценарий (2)

Отметим, что объект C посылает сообщение z(r) в окружение только при условии, что значение выражения r>0 истинно. Для экземпляра C альтернативные последовательности событий могут быть заданы на других диаграмах при помощи локального условия с тем же именем и новым выражением. В случеа, если альтернативная последовательность включает в себя взаимодействие с другими экземплярами, необходимо использовать глобальное состояние.

На рис. 11 приведен пример глобального потока данных между различными сценариями. На рис. 11 параметры сообщения *value* возвращаемые сценарием *pop* в качестве реакции на сообщение *get* зависят от событий в сценарии *push*. Подразумевается, что операции push и pop представляют собой процедуры на языке SDL, реализующие сообтветствующие стековые операции.

Основное назначение расширений языка диаграмм взаимодействия по работе с данными состоит в том, чтобы обеспечить возможность более точного определения функциональных требований на этапе сбора и анализа требований. Вместе с тем, те же самые расширения превращают язык диаграмм взаимодействия в мощное средство проектирования, пригодное для использования на более поздних этапах разработки (крупноблочное проектирование и детальное проектирование).

4.2. Алгоритм синтеза

Алгоритм синтеза состоит из следующих шагов [8]:

1. интеграция HMSC модели;
2. построение *MSC срезов;*
3. детерминизация MSC срезов;
4. минимизация MSC срезов;
5. генерация SDL поведения;
6. генерация SDL структуры.

Ключевым понятием алгоритма синтеза является понятие событийного автомата. **Событийный автомат** представляет собой конечный автомат, соответствующий единственному объекту диаграммы взаимодействия, так что входными символами автомата являются события данного объекта. Мы различаем три типа MSC событий: входные события, активные события и пустое событие.

- Пустое событие представляет собой вырожденное событие, добавленное для упрощения описания алгоритма синтеза.
- Входные события (требуют синхронизации с другими объектами, решение принимается другим объектом)
  - Прием сообщения *in(i,m)*
  - Срабатывание таймера *timeout(t).*
- Активные события (не требуют синхронизации с другими объектами, решения принимаются локально данным объектом)
  - Посылка сообщения *out(i,m)*
  - Действие *action(a)*
  - Установка таймера *set(t,d)*
  - Сброс таймера *reset(t)*
  - Остановка *stop*
  - Локальное условие *check(c).*

Наш алгоритм синтеза осуществляет построение событийных автоматов специального вида, так называемые MSC срезы. По определению, **MSC срез** (по данному объекту) представляет собой событийный автомат, генерирующий все

трассы событий для данного объекта. Для построения MSC среза используется следующий алгоритм:

1. начальные состояния событийного автомата совпадают с символами HMSC графа; все переходы событийного автомата помечены пустыми событиями;
2. для каждого базового MSC строится цепочка состояний, соответствующая последовательности событий для объекта, по которому производится срез;
3. каждая ссылка на базовую MSC из HMSC графа заменяется на цепочку состояний для этой базовой MSC;
4. стартовое состояние событийного автомата соответствует стартовому символу HMSC.

Иллюстрация работы алгоритма приведена на Рис. 13.

4.3. Пример

Проиллюстрируем работу алгоритма синтеза на небольшом примере, приведенном на рис. 12. MSC модель на рис. 12 состоит из двух сценариев Wait and Reply. Мы используем правило последовательной композиции. Объект R (получатель) соответствует системному актору, объект S (отправитель) соответствует внешнему актору.

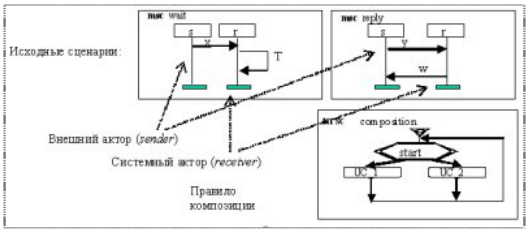

Рис. 12. Пример работы алгоритма синтеза

Отправитель S является инициатором обоих сценариев. Сценарий `wait` начинается с посылки объектом S сообщения X. Согласно данному сценарию, объект получатель R должен в этом случае «заснуть» на некоторый интервал времени. Сценарий `reply` начинается с посылки объектом-отправителем S сообщения Y. Согласно данному сценарию, объект-получатель R должен послать сообщениеW.

На Рис. 13 показаны отдельные шаги алгоритма построения MSC среза для объекта. Первый шаг алгоритма строит начальный событийный автомат по HMSC графу. Затем, строится событийный автомат для объекта S на диаграмме MSC wait. Данный автомат состоит из единственного перехода, помеченного активным событием *out(x,r).* На третьем шаге алгоритма, построенный автомат замещает соответствующий переход в начальном автомате. Следующие два шага обновляют начальный автомат на основе информации из диаграммы MSC reply для объекта S. Окончательный MSC срез для объекта S представлен в правом нижнем углу рис. 13.

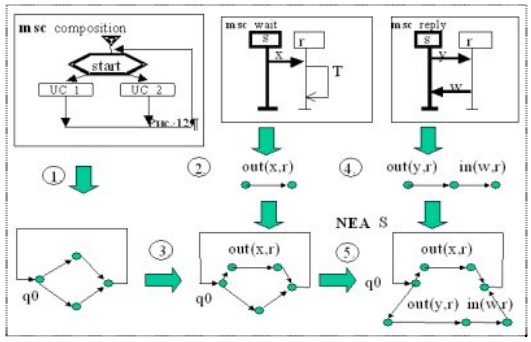

Рис. 13. Отдельные шаги алгоритма

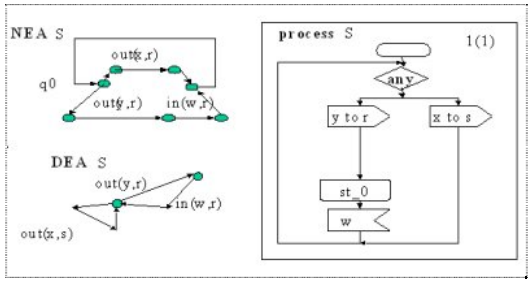

Рис. 14. Дальнейшие шаги алгоритма синтеза

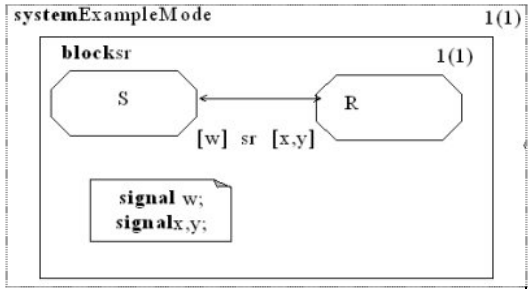

Рис. 15. Сгенерированная структура на языке SDL для примера на рис. 12

Рис. 14 показывает дальнейшие шаги алгоритма синтеза. Недетерминированный MSC срез для объекта S детерминизируется (с

смысле событийных автоматов) и минимизируется. На основе полученного событийного автомата DEA S генерируется SDL граф, описывающий поведение соответствующего SDL процесса. Рис. 15 показывает сгенерированную структуру на языке SDL для примера на рис. 12.

5. Сборка и валидация архитектуры

В данном разделе рассматриваются некоторые вопросы поддержки детализации моделей требований на языке SDL и доведение их до проектных моделей.

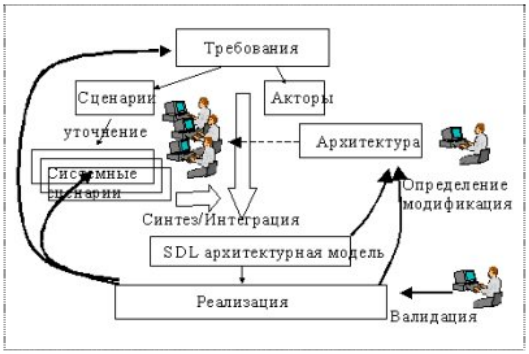

Рис. 16. Этап системного анализа

Методика ускоренной разработки программного обеспечения предполагает использование автоматического синтеза SDL спецификаций по сценариям на этапе системного анализа. На данном этапе происходит определение архитектуры проектируемой системы и разработка сценариев для каждого архитектурного компонента (рис. 16). Входная информация на данном этапе представлена в виде набора так называемых системных сценариев. Обычно, системный сценарий является проекцией некоторого сценария использования системы, построенного на этапе анализа требований. Структурная информация, присутствующая в системных сценариях, состоит из множества внешних акторов и множества архитектурных компонентов (представленных как отдельные объекты на диаграммах взаимодействия). Функциональная информация представлена в виде сценариев типичного взаимодействия между внешними акторами и архитектурными компонентами, а так же между компонентами, при этом цель системного сценария остается такой же, как в соответствующем исходном сценарии. Дополнительная функциональная информация может быть представлена в виде потоков данных по системным сценариям. По этой информации может быть синтезирована так называемая синтетическая архитектурная модель (CAM). Валидация синтетической архитектурной модели обеспечивает своевременную обратную связь как для разработчиков системных сценариев, так и для разработчиков архитектурной модели (рис. 16).

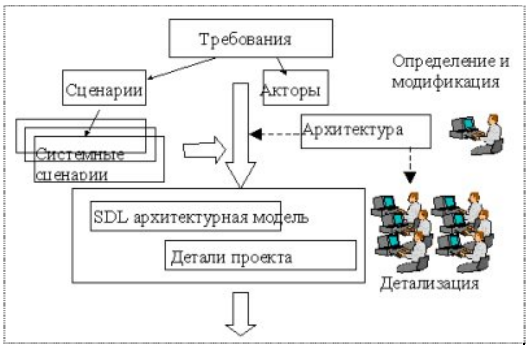

Рис. 17. Детализация архитектурной модели

В соответствии с алгоритмом синтеза, синтезированная архитектурная модель будет воспроизводить архитектурные компоненты, присутствующие в сценариях, синтезировать поведение каждого архитектурного компонента по всем имеющимся системным сценариям, и интегрировать модель в единое целое. Автоматически восстановленные связи между компонентами можно сравнить с предполагаемыми (описанными в архитектурной модели). Опыт использования синтезированных архитектурных моделей показывает, что они оказываются полезны для выявления дефектов этапа системного анализа. Описанный в данном разделе шаг метоики является прямым продолжением валидации требований.

Следующим шагом должна стать постепенная детализация синтезированной архитектурной модели, наполнение ее последующими проектными решениями (рис. 17). В качестве одного из способов детализации можно использовать вертикальную декомпозицию исходных MSC моделей (рис. 18). В сценарии reply из примера на рис. 12 объект R представлен в виде набора из трех объектов нижнего уровня: R1, R2, R3. Потоки сообщений между

объектами нижнего уровня на диаграмме декомпозиции должны быть совместимы с потоками сообщений на родительской диаграмме. Диаграмма декомпозиции задает два варианта поведения, которые не были наблюдаемы на родительской диаграмме. Альтернативы задаются при помощи состояния с одним и тем же именем. Синтезированная архитектурная модель для данного примера приведена на рис. 19

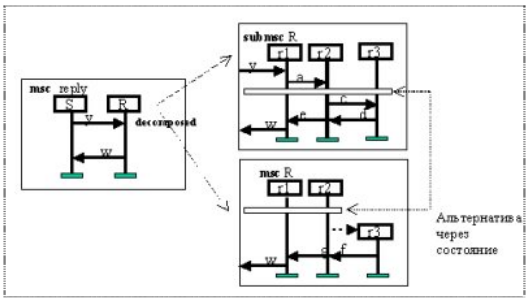

Рис. 18. Вертикальная декомпозиция

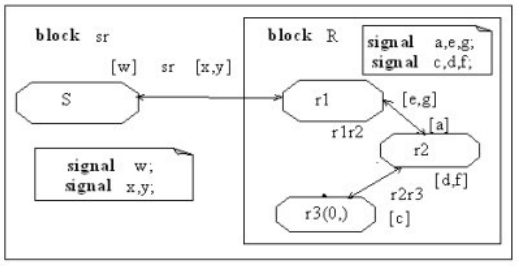

Рис. 19. Синтезированная архитектурная модель

6. Адаптивная генерация кода по SDL спецификациям

В данном разделе мы описываем наш опыт разработки генераторов кода для языка SDL для промышленного использования [1,3,4,6,9,10,11,31]. В начале раздела мы ставим проблему адаптации сгенерированного кода. Затем мы сравниваем несколько подходов к адаптивной генерации кода и перечисляем основные препятствия, затрудняющие адаптацию сгенерированного кода. Затем мы описываем понятие декларативных отображений [11,31] как одну из перспективных возможностей для генерации адаптивного кода.

Автоматическая генерация кода по SDL спецификациям является гораздо более распространенной, чем может показаться на первый взгляд. Сгенерированый код используется в большинстве коммерческих SDL CASE-систем для исполнения, валидации и генерации тестов. Некоторые SDL системы используют сгенерированный код для оценки производительности системы. Вместе с тем, наибольшее преимущество, связанное с автоматической генерацией кода, заключается в возможности использования сгенерированного кода непосредственно в продуктах [3,29]. Однако, большинство промышленных проектов избегают использование существующих кодогенераторов на этапе реализации [20].

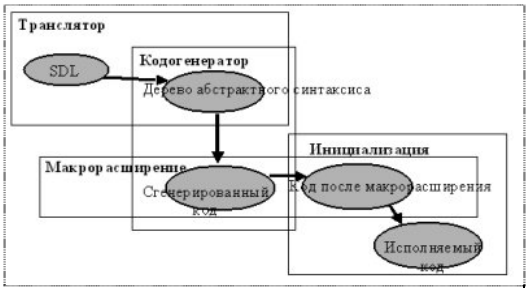

Рис. 20. Основные преобразования при генерации кода по SDL спецификациям

Рис. 20 представляет основные преобразования производимые при генерации кода по SDL спецификациям.

Использование сгенерированного кода в промышленных проектах сталкивается в целым рядом трудностей [3,29]:

- использование необходимого целевого языка кодогенерации (CHILL, C, C++, Java, и т.д.);
- качество сгенерированного кода [6,10,11]:
  - размер сгенерированного кода,
  - эффективность сгенерированного кода,
  - читабельность сгенерированного кода;
- программные интерфейсы к сгенерированному коду [3];
- интегрирование сгенерированного кода с унаследованным кодом;
- непосредственное использование кодовых вставок на целевом языке в сгенерированном коде [3];
- использование различных отображений для разных частей системы (оптимизации);
- сборки и интеграция системы, использующей сгенированный код.

Наш опыт разработки кодогенераторов

для промышленных проектов сдидетельствует о том, что каждый проект выдвигает уникальные требования к сгенерированному коду. Это означает, кодогенератор и сгенерированный кода должны обладать свойством адаптивности к требованиям конкретного проекта, кодогенератор должен позволять изменять решения о структуре и поведении сгенерированного кода. Согласно [3] идеальной является ситуация, когда кодогенератор является дополнительным «сотрудником» проекта, т.е. полностью вписывается в требования конкретного проекта.

В настоящее время несколько исследовательских групп ведут разработку различных аспектов адаптивной кодогенерации. Существующие подходы к адаптивной кодогенерации включают в себя следующее:

- система поддержки этапа выполнения (библиотека или объектно-ориентированный каркас);
- использование макросов в сгенерированной коде;
- настраиваемый кодогенератор;
- поддержка кодовых вставок в спецкомментариях;
- аннотации этапа кодогенерации;
- декларативные отображения.

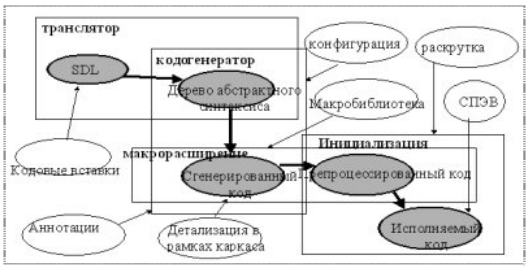

Рис. 21. Адаптация сгенерированного кода может осуществляться на различных этапах

Основной трудностью кодогенерации является разработка соответствующего *отображения* [6,11]. **Отображением** называется согласованный набор решений по представлению исполняемых и структурных конструкций исходного языка, таких что результирующая программа на целевом языке кодогенерации оказывается синтаксически и семантически правильной с точки зрения целевого языка. Существует целый ряд проблем, связанных с отображениями, которые существенно сокращают пространство проектных решений при разработке адаптивного кодогенератора:

Исполняемые конструкции исходного языка достаточно часто имеют семантически близкие эквиваленты в целевом языке, что позволяет отображать их напрямую в эти конструкции (например, циклы, условные выражения, присваивания и т.д.).

- Семантически далекие исполняемые конструкции исходного языка могут быть смоделированы группами исполняемых конструкций целевого языка (возможно с использованием некоторых структурных конструкций). Часто такие группы могут быть инкапсулированы в библиотеку поддержки этапа выполнения.
- Некоторые структурные конструкции могут быть представлены семантически близкими эквивалентами (например, объекты, процедуры модули, пакеты и т.д.)
- Некоторые структурные конструкции требуют глубокой трансформации во время трансляционных преобразований, выполняемых при кодогенерации (что может существенно снижать возможности последующей адаптации
- Большинство структурных конструкций могут быть проинтерпретированы на этапе исполнения (что существенно неэффективно).
- Некоторые структурные конструкции могут быть представлены группами структурных (а также с добавлением некоторого количества исполнимых) конструкций, однако это незбежно приводит к явлению фрагментации сгенерированного кода [6] и отрицательно сказывается на адаптации сгенрированного кода.

Следующие факторы снижают возможность адаптации сгенрированного кода:

- решения, связанные с отображением исходного языка в целевой, не локализованы;
- решения, связанные с отображением, сильно взаимосвязаны;
- пространство возможных отображений оказывается весьма разреженным из-за семантических и синтаксических ограничений целевого языка (несмотря на то, что для каждой отдельно взятой

конструкции исходного языка может существовать несколько вариантов отображения в целевой язык);
- решения, связанные с отображением, как правило, невозможно изменять на более поздних этапах преобразований, составляющих процесс кодогенерации.

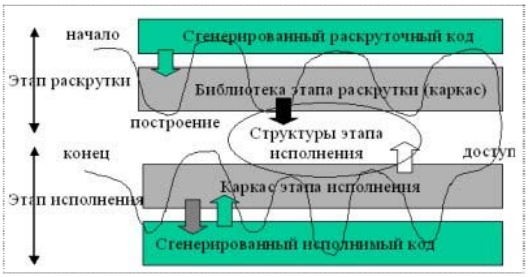

Рис. 22. Концепция декларативных отображений

В [6,11] нами была предложена концепция декларативных отображений как перспективное направление генерации адаптивного кода по SDL спецификаций (рис. 22). Следующие особенности составляют определение декларативного отображения:
- решения относительно представления структурных конструкций исходного языка откладываются на самый поздний из возможных этапов, т.е. на этап исполнения сгенерированной программы;
- сгенерированный код содержит «декларации», представляющие собой непроцедурные указания того, какие конструкции исходного языка используются в конкретной программе;
- «декларации» исполняются на так называемом этапе раскрутки сгенерированной программы (являющегося начальной стадией этапа исполнения;
- поддержка этапа раскрутки представляет собой библиотеку, которая определяет то, какие структуры данных этапа исполнения будут построены для каждой из конструкций исходного языка;
- системы поддержки этапа исполнения представляет собой каркас для сгенерированных «исполнимых» конструкций и структур данных, построенных на этапе раскрутки.

Автоматическая генерация кода по SDL спецификациям является очевидным способом ускорения разработки программного обеспечения. Однако, как было продемонстрировано выше, требуются значительные усилия для обеспечения интеграции сгенерированного кода с остальными частями окончательной программной системы в реальных промышленных проектах. Мы считаем, что разработка адаптивных кодогенераторов является весьма актуальной для более широкого внедрения автоматической кодогенерации. По нашему опыту, не существует простых и универсальных решений задачи адаптивной кодогенерации. Скорее всего, успешной окажется комбинация подходов, описанных в данном разделе, включая концепцию декларативных отображений.

7. Восстановление SDL спецификаций по исходным текстам унаследованных систем

CASE-системы нового поколения, использующие SDL, приводят к значительным улучшениям качества процесса разработки программного обеспечения, производительности труда и времени выхода окончательного продекта на рынок. Однако для более широкого внедрения CASE-систем в телекоммуникационном секторе необходимо разработать эффективные методы интеграции компонентов, производимых автоматизированно, с уже существующими, так называемыми *унаследованными* компонентами. Обычно, унаследованное базовое программное обеспечение разрабатывалось с применением смеси различных языков высокого и низкого уровней, различных архитектур и стилей. Особенностью унаследованных систем является плохая документированность проектных решений. До недавнего времени, все жто составляло существенный «барьер» для внедрения формальных методов [5,13].

Для преодоления «барьера» унаследованного кода необходимо разрабатывать автоматические методы восстановления формальных спецификаций по исходным текстам программ. Наличие эффективных методов восстановления SDL моделей позволит достичь следующие цели:
- понимание структуры и поведения унаследованного программного обеспечения. Известно, что исполнение SDL модели дает весьма наглядные

результаты и не требует использования дорогостоящего спецоборудования (рис. 23);
- автоматизированная генерация тестов для унаследованного программного обеспечения (рис. 23);
- анализ и валидация формальных спецификаций нового программного обеспечения, разрабатываемого на базовой платформе (возможно, с использование формальных методов) (рис. 24);
- автоматическая генерация тестов для нового программного обеспечения, разрабатываемого на базовой платформе (рис. 24);
- автоматическая генерация кода для нового программного (рис. 24). Для интеграции генерированного кода с унаследованными компонентами требуется использование адаптивных кодогенераторов (разд. 6).

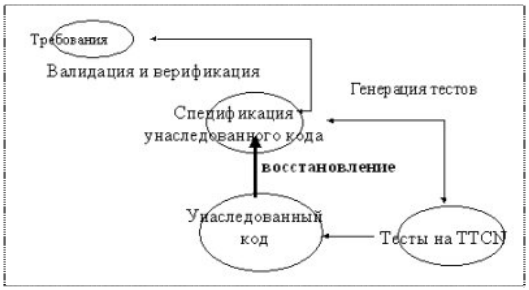

Рис. 23. Унаследованное программное обеспечение

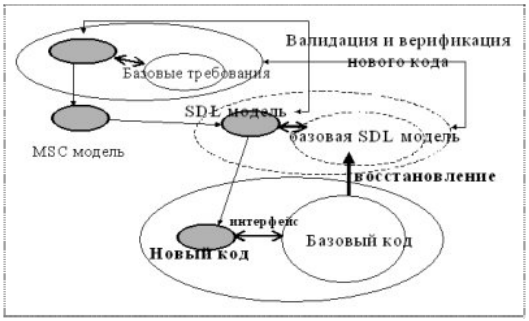

Рис. 24. Новое программное обеспечение

В данной статье описывается методика динамического восстановления формальных SDL спецификаций унаследованного телекоммуникационного программного обеспечения по сценариям.

Особенности нашего подхода состоят в следующем:

- установка семантических датчиков [5] в исходные тексты унаследованных компонентов в соответствии с некоторой стратерией инструментирования, выработанной на основе архитектурного анализа кода;
- выбор представительных сценариев из регрессионных тестов и других источников;
- исполнение сценариев на инструментированном коде унаследованной системы с последующей генерацией трасс в формате MSC;
- синтез SDL спецификации с использованием Московского Синтезатора по набору MSC [8].

Этот процесс повторяется до тех пор, пока восстановленная SDL спецификация не станет удовлетворять определенным критериям [5]. Полученная SDL модель используется для оценки и улучшения покрытия набора регрессионных тестов.

7.1 Обзор методики восстановления

Динамический подход к восстановлению SDL спецификации унаследованного компонента по сценариям представляет собой процесс синтеза модели по трассам, получаемых за счет исполнения инструментированной системы (Рис. 25). Исполнение системы управляется набором унаследованных тестов [5].

Автоматический синтез SDL спецификаций по MSC является ключевой технологией предлагаемого динамического подхода к восстановлению спецификаций. В предыдущих разделах статьи, синтез SDL по MSC рассматривался исключительно как техника прамой инженерии. В динамическом подходе мы используем двойственность нотации MSC, пригодной как для для описания требований, так и для описания трасс.

Альтернативным подходом в восстановлению SDL спецификаций является так называемый прямой подход [2,5]. Сравнение подходов приводится на Рис. 25. Прямое восстановление спецификаций происходит непосредственно за счет проведения некоторого семантически корректного трансляционного преобразования [2]. Интересный подход к восстановлению спецификаций представлен в [21]. Мы называем жтот подход частичным восстановлением. При частичном

восстановлении выполняется выборочное преобразование управляетмое так называемой картой предолжений исходной программы (рис. 25). Записи в карте состояний содержат все предолжения из исходной программы (в некоторой канонической форме. В катре предолжений вручную задается правило перевода всех вхождений предложений данного типа. Каркас модели восстанавливается автоматически.

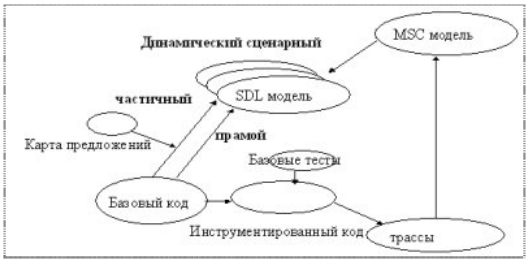

Рис. 25. Выборочное преобразование

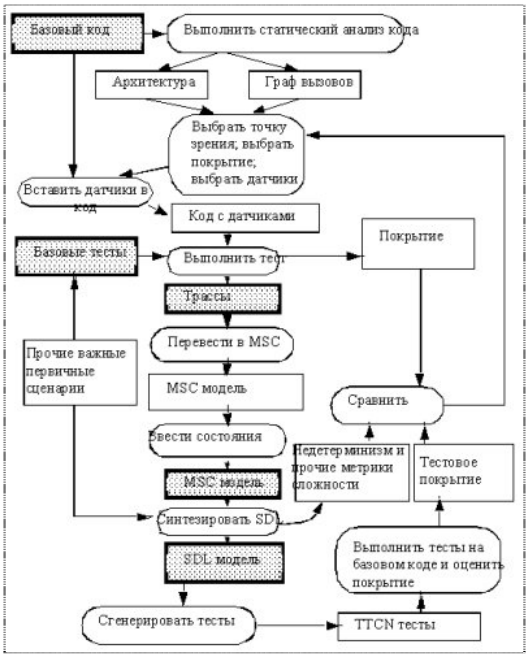

Рис. 26. Обзор методики восстановления

Описываемый подход к восстановлению SDL спецификаций представляет собой итеративный процесс, состоящий их следующих четырех этапов:

1. подготовка;
2. динамический сбор трасс;
3. синтез SDL модели;
4. исследование SDL модели.

Каждый этап состоит из нескольких шагов. Итерации осуществляются под упраавлением критериев завершения, проверяемых на последнем этапе. Обзор методики восстановления представлен на рис. 26.

7.3 Детальное описание методики восстановления

7.3.1. Фаза подготовки

Целью данного этапа является разработка стратегии установки датчиков и выбор набора сценариев, под управлением которых будет выполняться проинструментированная система.

Шаг 1. **Анализ кода.** Целью данного шага является определение стратегии установки датчиков. На данном шаге применяются известные методы статического структурного анализа. Две модели унаследованного компонента оказываются полезными для определения датчиков – архитектурная модель системы (основные компоненты и их связи) и граф вызовов системы [7]. Граф вызовов системы должен показывать внешние интерфейсы системы (обычно – системные вызовы или ассемблерные вставки).

Шаг 2. **Выбор точки зрения моделирования**. Динамический подход к восстановлению спецификаций может применяться для обратной инжеенрии и генерации тестов исходя из трех различных точек зрения: «черный ящик» (окружение), «белый ящик» (базовый код) или «серый ящик» (взаимодействие подсистем). Точка зрения моделирования существенным образом овлияет на структуру получаемой SDL модели. Она также влияет на уровень детализации трасс и, тем самым, на объем работы по адаптации автоматически сгенерированных тестов [19] (см. также шаг 11).

Шаг 3. **Выбор критерия покрытия и набора датчиков**. На данном шаге окончательно определяется стратегий установки датчиков в исходный текст системы (определяются конкретные места в исходном коде, куда будут установлены датчики и определеляется формат выдачи каждого датчика). Стратегия установки датчиков в большой степени определяется критерием покрытия. Внешний интерфейс трассируемой части системы определяется в терминах точек на архитектурной модели и графе вызовов таким образом, что датчики

регистрируют прохождение информации через выбранную границу и записывают ее в желаемом формате.

Шаг 4. **Выбор первичных сценариев.** Динамическое исполнение системы и сбор трасс управляется некоторым набором тестов. Мы предлагаем использовать для первой итерации (унаследованный) набор регрессионных тестов

Итеративный процесс должен начинаться с регрессионных тестов. Такие тесты обычно представляют из себя смесь тестов сертификации (обычно только штатное поведение, низкая производительность), первичных сценариев (низкая производительность) и некоторых наиболее важных вторичных сценариев (средняя или высокая производительность). На последующих итерациях для улучшения покрытия добавляются новые функциональные сценарии. По мере совершенствования модели, она может быть использована для автоматической генерации тестов на языке TTCN согласно методикам, хорошо известным по литературе [19,20]. На данной фазе процесса мы становимся больше заинтересованы в новых вторичных, высокопроизводительных сценариях. Производительность сценариев обсуждалась в разд. 3.

7.3.2 Получение трасс

Целью данного этапа является получение набора трасс, соответствующих выбранной стратегии установки датчиков и выбранному набору сценариев.

Шаг 5. **Инструментирование системы.** Для выполнения данного шага требуется создать инфраструктуру, позволяющую проводить инструментирование и исполнение унаследованной системы. Датчики должны устанавливаться в соответствии с выбранной стратегией установки датчиков. Идеальной является ситуация, когда инструментирование выполняется автоматически.

Шаг 6. **Запуск инструментированного кода.** Задачей данного шага является сборка и запуск проинструментированной унаследованной системы. Для исполнения системы может быть использовано целевое спецоборудование или его эмуляторы. Результатом шага является набор трасс. Дополнительным результатом данного шага является информация о покрытии системы выбранными сценариями.

7.3.3 Синтез SDL модели

Даный этап является ключевым для нашего динамического подхода. Целью этапа является синтез SDL модели.

Шаг 7. **Перевод трасс в формат MSC.** Данный шаг был введен в методику для разделения двух задач: динамического сбора трасс по унаследованному коду и синтеза SDL моделей по сценариям. Задачей данного шага является простое преобразование трасс в снтаксис языка диагармм взаимодействия. Данный шаг определяется различиями между форматом трасс (выбираемый на шаге инструментирования) и форматом MSC, поддерживаемого Московским Синтезатором.

Шаг 8. **Введение состояний в MSC.** Данный шаг выполняет процесс «абстакции» представленный на Рис. 26. Целью шага является выявление последовательностей обменов сообщениями соответствующих транзакциям в сценариях использования. На основе этой информации линейные диагарммы взаимодействия, соответствующие трассам, преобразуются в MSC модель. Сущность данного преобразования заключается во введении состояний в те точки трасс, где возможны повторы или альтернативы. Введение состояний «обогащает» входные MSC модели, что позволяет синтезировать более интересные SDL модели.

Шаг 9. **Синтез SDL моделей**. Данный шаг выполняется автоматически Московским Синтезатором (MOST-SDL) [5]. Техника синтеза была описана в разделе 4. Результатами данного шага являются: 1) синтезированная SDL модель и 2) некоторые метрики сложности синтезированной модели – число состояний в модели и метрика недетерминированности синтезированной модели. Последняя используется как неявный критерий завершения итераций процесса восстановления спецификации. Смысл данной метрики заключается в следующем: недетерминизм синтезированной модели возникает каждый раз, когда два или более сценариев описывают различное поведение для одного и того же принимаемого сообщения. На практике, это обычно является следствием зависимости поведения от истории выполнения, а трассы не содержат достаточно информации. Большие занчения метрики недетерминированности синтезированной SDL модели требуют проведения дополнительных итераций.

7.3.4 Исследование SDL модели

Целью данного этапа является обновление исходного набора тестов за счет автоматической генерации новых тестов по синтезированной SDL спецификации, проверка критериев окончания итераций путем сравнения инструментального покрытия и тестового покрытия, а также метрики недетерминированности синтезированной модели.

Шаг 10. **Генерация TTCN тестов.** Одной из основных целей восстановления SDL модели является использование ее автоматической генерации тестов. Техника генерации тестов по SDL моделям описаны в [19,20].

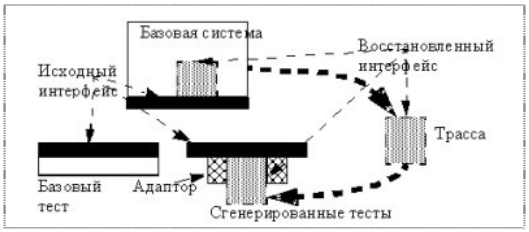

Рис. 27. Адаптация интерфейсов

Шаг 11. **Исполнение** сгенерированных **тестов на унаследованной системе и оценка покрытия.** Исполнение сгенерированных тестов может потребовать дополнительной адаптации, т.е. преобразования абстрактных интерфейсов в исходные [19]. Адаптация интерфейсов представлена на рис. 27. Трассы и соответствующие им автоматически сгенерированные тесты используют так называемые абстрактные интерфейсы, соответствующие выбранной стратегии установки датчиков в исходный код унаследованного компонента. Абстрактный интерфейс использует некоторые внутренние функции («серый ящик») внутри системы, которые не обязательно доступны из окружения унаследованной системы. На Рис. 27 это показано как серая полость внутри прямоугольника, представляющего унаследованную систему. Вместе с тем, регрессионные тесты используют так называемые внешние интерфейсы системы («черный ящик»). На рис. 27 это представлено в виде нижнего черного прямоугольника на изображении унаследованной системы. Назначение адаптора (заштрихованный прямоугольник на рис. 27) состоит в том, чтобы преобразовывать абстрактные интерфейсы во внешние. Наш опыт показывает, что должен достигаться некоторый компромисс между «глубиной» абстрактного интерфейса и простотой реализации адаптора. Абстрактный интерфейс на логическом уровне существенно упрощает трассы и позволяет получать более осмысленные синтезированные SDL модели. В некоторых случаях оказывается удобным реализовать специальную поддержку доступа к внутренним функциям для упрощения реализации адаптора.

Шаг 12. **Критерий завершения.** Целью данного шага является проверка адекватности восстановленного поведения унаследованной системы в синтезированной модели. Как уже было отмечено ранее, методика восстановления предполагает проведение итераций. Неадекватное восстановление может быть вызвано следующими факторами: 1) некоторый важный сценарий не был исполнен при сборе трасс; 2) неправлильно был выбран абстрактный интерфейс при разработке стратегии установки датчиков (например, пропущен датчик, или датчик установлен неверно).

По нашему опыту, неправильно установленные датчики выявляются при изучении инструментального покрытия. Пропущенные датчики на внешних интерфейсах приводят к большим значениям метрики недетерминированности синтезированной системы. Пропущенные датчики на выходных интерфейсах выявляются при изучении покрытия сгенерированных тестов. Такми образом, инструментальное покрытие, метрика недетерминированности и покрытие синтезированных тестов совместно являются удовлетворительным критерием адекватности синтезированной модели.

8. Выводы и направления дальнейших исследований

Мы описали некоторый опыт в построении инструментальных систем нового поколения, основанных на использовании формальных методов и направленных на достижение практических улучшений процесса разработки программного обеспечения в телекоммуникационном секторе. Нами была описана методика ускоренной разработки программного обеспечения, охватывающая этапы составления спецификаций, проектирования,

тестирования и восстановления спецификаций телекоммуникационного программного обеспечения. Особенностями предлагаемой методики является применение формальных методов для описания моделей программного обеспечения на ранних этапах разработки, а также автоматизированное восстановления формальных спецификаций программного обеспечения на поздних этапах (этапе сопровождения). Методика основана на использовании стандартных формальных языков спецификации, широко распространенных в телекоммуникационном секторе: язык Спецификаций и Описаний (SDL), язык диаграмм взаимодействия (MSC), язык описания тестов (TTCN) и язык описания данных (ASN.1).

В статье были описаны следующие ключевые шаги методики:

- описание требований к проектируемой системе в виде сценариев, формализованных на языке диаграмм взаимодействия;
- высокопроизводительная валидация требований с использованием исполняемых SDL моделей;
- синтез исполнимых моделей требований на языке SDL по сценариям на языке MSC;
- детализация моделей требование на языке SDL и доведение их до проектных моделей;
- адаптивная генерация кода по спецификациям на языке SDL;
- автоматическое восстановление SDL моделей по исходным текстам программ унаследованного программного обеспечения.

Одной из главных задач нашего исследования является снижение барьеров для более широкого внедрения формальных методов на очень ранних и очень поздних этапах разработки.

Методика ускоренной разработки программного обеспечения предполагает увеличение начальных этапов разработки. Утверждается, что увеличение продолжительности начальных этапов примерно на 17%, требуемых для составление и валидацию формальной модели требований и генерацию тестов на раннх этапах, может привести в общему сокращению сроков разработки на 20-30%. Мы разрабатываем инструментальные средства нового поколения, поддерживающие нашу методику и позволяющие реализовать указанные сокращения сроков разработки.

В настоящее время мы продолжаем выполнение исследований по программе, описанной в данной статье. Продолжается определение методики ускоренной разработки и уточнение отдельных ее шагов, а также определение открытой архитектуры, состоящей из языков и соответствующих инструментальных средств поддерживающх предлагаемую методику и автоматизирующих ее насколько возможно. Например, предметом нашего изучения являются методы быстрого построения тестов по SDL моделям и методы архитектурного анализа унаследованного телекоммуникационного программного обеспечения. Другие направления дальнейших исследований отдела инструментальных системе поддержки разработки включают в себя выполнение «пилотных проектов», имеющих целью оценить практическую применимость различных подходов к восстановлению формальных спецификаций, а также эффективность совместного использования методов прямой и обратной инженерии для ускорения сроков разработки и сокращения времени выхода продукта и его последующих версий на рынок.

Литература

1. N. Mansurov, A. Kalinov, A. Ragozin, A. Chernov (1995), Design Issues of RASTA SDL-92 Translator, in Proc. of the 7-th SDL Forum, Oslo, Norway, 26-29 September, 1995, Elsevier Science Publishers B.V. (North-Holland), pp. 165-174
2. N. Mansurov, E. Laskavaya, A. Ragozin, A. Chernov, Об одном подходе к использованию языков SDL и MSC для обратной инженерии, в сб. Вопросы Кибернетики, Приложения Системного Программирования, N. 3, Москва,
3. N. Mansurov; A. Ragozin; A. Chernov; I. Mansurov: “Industrial strength code generation from SDL”, in A. Cavalli, A. Sarma (Eds.) SDL’97: TIME FOR TESTING – SDL, MSC and Trends, Proc. Of the 8-th SDL Forum, Evry, France, 23-26 September, 1997, , Elsevier Science Publishers B. V. (North-Holland), pp. 415—430.
4. N. Mansurov, A. Ragozin, A. Chernov, I.Mansurov, Tool suppport for algebraic Specifications of Data in SDL-92, in Proc. Formal Description Techniques IX, Kaiserslautern, Germany, 8-11 October 1996, Chapman & Hall, p 61-76.
5. N. Mansurov, R. Probert, Dynamic scenario-based approach to re-engineering of legacy telecommunication software, in Proc. 9th SDL

Forum, Montreal, Canada, June 21-26, 1999, Elsevier Science Publishers B.V. (North-Holland).
6. N. Mansurov, A. Ragozin, Generating readable programs from SDL, in Proc. SAM'98 workshop, 29 June-1 July, 1998, Humboldt University, Berlin, 1998.
7. N. Rajala, D. Campara, N. Mansurov, inSight Reverse Engineering CASE Tool, in Proc. of the ICSE'99, Los Angeles, USA, 1998.
8. N. Mansurov, D. Zhukov, Automatic synthesis of SDL models in Use Case Methodology, in Proc. 9th SDL Forum, Montreal, Canada, June 21-26, 1999, Elsevier Science Publishers B.V. (North-Holland).
9. N.Mansurov, A. Chernov, Генерация исполняемого кода по алгебраическим спецификациям, в сб. Вопросы Кибернетики, Приложения Системного Программирования, N. 2, Москва, 1996
10. [N.Mansurov, A. Ragozin, Генерация кода с простой и наглядной структурой по спецификациям на языке SDL, в сб. Вопросы Кибернетики, Приложения Системного Программирования, N. 3, Москва, 1997
11. N. Mansurov, A.Ragozin, Using declarative mappings for automatic code generation from SDL and ASN.1, in Proc. 9th SDL Forum, Montreal, Canada, June 21-26, 1999, Elsevier Science Publishers B.V. (North-Holland).
12. N. Mansurov, O. Majlingova, Formal specification methodologies: MSC and SDL languages (lecture notes and tutorial), Moscow State University, Department of Computational Mathematics and Cybernetics, 1998 (in Russian).
13. R. Probert, N. Mansurov, Improving time-to-market using SDL tools and techniques (tutorial), Proc. 9th SDL Forum, Montreal, Canada, June 21-26, 1999.
14. [ITU-T (1993), CCITT Specification and Description Language (SDL), ITU-T, June 1994
15. Z.120 (1996) CCITT Message Sequence Charts (MSC), ITU-T, June 1992
16. R.L. Probert, O.Monkewich, TTCN: the international notation for specifying tests for communications systems, Computer Networks, Vol. 23, No. 5, 1992, pp. 417-738
17. ISO/IEC 8824: Specification of Abstract Syntax Notation One (ASN.1) (1989).
18. Telelogic (1998), Telelogic ORCA and SDT 3.3, Telelogic AB, Box 4128, S-203 12 Malmoe, Sweden, 1998
19. Anders Ek, Jens Grabowski, Dieter Hogrefe, Richard Jerome, Beat Koch, Michael Schmitt, Towards the Industrial Use of Validation Techniques and Automatic Test Generation Methods for SDL Specifications, in Proc. of the 8-th SDL Forum, Evry, France, 23-26 September, 1997, Elsevier Science Publishers B.V. (North-Holland), pp. 245-261
20. R. Probert, H. Ural, A. Williams, J. Li, R. Plackowski, Experience with rapid generation of functional tests using MSCs, SDL, and TTCN, submitted to Special Issue of Formal Descriptions Techniques in Practice of Computer Communications, 1999
21. G. Holzmann, Formal Methods for Early Fault Detection, (invited paper) in 4th Int. School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems, September 1996, Uppsala, Sweden.
22. I. Jacobson, M. Christerson., P. Jonsson, G. Overgaard, Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, Reading, MA, 1992.
23. G. Robert, F. Khendek, P. Grogono, Deriving an SDL specification with a given architecture from a set of MSCs, in Proc. of the 8-th SDL Forum, Evry, France, 23-26 September, 1997, Elsevier Science Publishers B.V. (North-Holland), pp. 197-212
24. R.Tuok, L. Logrippo, Formal specification and use case generation for a mobile telephony system, Computer Networks and ISDN Systems, 30 (1998), pp. 1045-1063.
25. M. Andersson, J. Bergstrand, Formalization of Use Cases with Message Sequence Charts, MSc Thesis, Lund Institute of Technology, May 1995
26. R. L. Probert, K. Saleh, Synthesis of communication protocols: survey and assessment, IEEE Transactions on Computers, 40(4), pp. 468-475, April 1991
27. S. Leue, L. Mehrmann, M. Rezai, Synthesizing ROOM Models from Message Sequence Chart Specifications, University of Waterloo, Technical Report 98-06, 1998
28. S. Some, R. Dssouli, and J. Vaucher, From scenarios to timed automata: Building specifications from user requirements, In Proc. 2nd Asia Pacific Software Engineering Conference, IEEE, December 1995.
29. R. Singh, J. Serviss, Code Generation using GEODE: A CASE Study, in Proc. of the 8-th SDL Forum, Evry, France, 23-26 September, 1997, Elsevier Science Publishers B.V. (North-Holland), pp. 539-550
30. J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modelling Language Reference Manual, Addison-Wesley, 1999
31. A. Ragozin, Automatic generation and execution of programs from SDL specifications, PhD thesis, Moscow State University, 1999