СОВМЕСТНОЕ ИСПОЛЬЗОВАНИЕ MSC И SDL МОДЕЛЕЙ ПРИ РАЗРАБОТКЕ СОБЫТИЙНО-ОРИЕНТИРОВАННЫХ СИСТЕМ

Диссертант:	Соколов В.В.
Научный руководитель:	д.ф.-м.н., проф. Терехов А.Н.
Ведущая организация:	Институт Системного Программирования РАН
Официальные оппоненты:	д.т.н., проф. Гольдштейн Б.С. к.ф.-м.н. Костин В.А.

Представляемая работа посвящена совместному использованию MSC и SDL моделей при разработке событийно-ориентированных систем.

Введение: современные системы

- Распределенность
- Сложность
- Параллелизм
- Встроенность и отказоустойчивость
- Различные производители

- Формальное описание протоколов взаимодействия
- Использование стандартов на протоколы

Выбранная тема является актуальной, поскольку современные системы обладают рядом параметров, из-за которых они состоят из множества взаимодействующих частей. Протоколы их взаимодействия являются достаточно сложными, поэтому их надо формализовывать. Части могут создаваться различными производителями, поэтому формализация должна следовать общепринятым стандартам.

Дополнение:

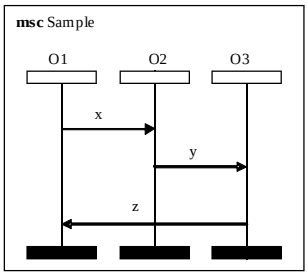
1. Рассказывается о современных системах и их характеристиках
   (то, что написано до черты)
   - распределенность – то, что современные системы теперь не состоят из одного компьютера, стоящего на столе, а это теперь комплексы различных устройств
   - сложность – большое количество выполняемых функций
   - параллелизм – в подобных системах процессы выполняются параллельно
   - встроенность – приложения везде – в автомашинах, бытовых приборах, аэропортах – поэтому встает вопрос отказоустойчивости
   - необходимость согласования ПО и аппаратуры различных производителей

Как следствие:

2. Описывается появление протоколов взаимодействия
3. Для их описания необходимы стандарты

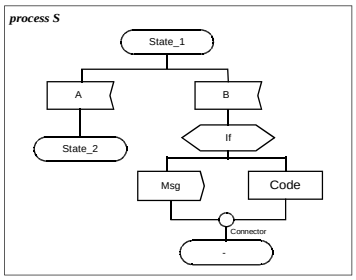
Введение: стандарты MSC и SDL

Message **S**equence **C**hart



ITU-T Recommendation Z.120

Specification and **D**escription **L**anguage



ITU-T Recommendation Z.100

Наиболее зрелыми являются стандарты MSC и SDL. Они пришли из телекоммуникационной области, на их языке описываются другие стандарты ITU-T. MSC описывает систему “в целом”, вертикальные линии обозначают отдельные объекты, время течет сверху вниз, горизонтальные линии изображают обмен сообщениями. Видна логика обмена сообщениями между различными объектами, логика принятия решения каждого из них скрыта. SDL является расширенной конечно-автоматной моделью, видна логика принятия решений каждым из объектов, но не видно их взаимодействие.

Дополнение:

1. Вкратце говорится про стандарты MSC и SDL

Пришли из телекоммуникационной области, поскольку в ней проблемы, перечисленные на предыдущем слайде, появились раньше.

Используются при создании телефонных станций, описывают различные протоколы взаимодействия.

2. Даются их характеристики

MSC – описание взаимодействия различных объектов с помощью сообщений. Взгляд со стороны. Видно, как взаимодействуют объекты, однако логика принятия решений каждого из объектов скрыта.

SDL – расширенная конечно-автоматная модель. Сочетает в себе блок-схемы и конечно-автоматную модель. Описывает внутреннюю логику каждого из объектов. Сценарии взаимодействия нескольких объектов не видны.

3. Выводы

- общим у данных стандартов является то, что они описывают взаимодействие с помощью сообщений
- используются оба, поскольку дают различный взгляд на систему.

Более того, есть информация, которая содержится только на «своей» модели

- обычно используются на различных этапах создания ПО
  -- MSC при проектировании
  -- SDL при программировании

Введение: Подходы и проблемы совместного использования MSC и SDL

Подходы:

- MSC и SDL независимы
- Генерация SDL по MSC
  - система описывается на MSC, SDL не редактируется
  - одноразовый синтез
  - инкрементальный подход
- Верификация
  - трассы (MSC на SDL)
  - состояния протокола (параллельное исполнение моделей)

Проблемы:

- Неуправляемость генерации
- Проблемы с верификацией при рассогласованиях моделей
- Слабость используемой MSC модели

4

Необходимость использования обоих стандартов диктуется технологическими требованиями. Поскольку они описывают одни и те же алгоритмы обмена сообщениями, то они должны использоваться согласованно. Независимость ведет к ошибкам. Для их согласования используется либо генерация SDL по MSC, либо верификация. Более детально про подклассы каждого из методов будет рассказано дальше в соответствующем разделе.

В данных методах есть ряд проблем:

• Генерация выступает как черный ящик, на вход которого подаются MSC диаграммы, а на выходе появляются SDL диаграммы. Они могут быть не удобны для разработчика.

• Верификация работает недостаточно хорошо при изменениях моделей.

• Существующий стандарт MSC не позволяет описать достаточно полные модели для использования их в генерации и верификации.

Дополнение:

1. Описываются существующие подходы по сочетанию SDL и MSC в одном средстве разработки:
   - • независимы – несогласованность моделей
   - • описание только на MSC – плохо, когда выкидывается SDL модель, лучше приспособленная для программирования
   - • одноразовый синтез – надо следить за последующим рассогласованием моделей
   - • инкрементальный подход – работает в очень малом количестве случаев
2. Верификация
   - • трассы – при данном подходе MSC трассы «накладываются» на SDL модель
   - • состояния протокола – параллельное исполнение двух моделей с определенной математикой, позволяющей свести бесконечное множество вариантов к некоему проверяемому классу
   - • *дополнительные возможности по поиску проблем <u>только</u> в одной модели*
3. Описываются проблемные места, для данных проблем далее предлагаются решения

Генерация: обзор подходов

Обобщенная модель генерации: MSC → FSM → SDL

где **FSM** (**F**inite **S**tate **M**achine) описывает поведение только одного объекта

Подходы и их оценка:

- Непосредственный (FSM модель не подвергается преобразованиям)
  - \+ в полученной SDL модели лучше узнаваема MSC модель
  - – невозможность синтеза в части случаев
  - – излишне громоздкая SDL модель в части случаев
- С преобразованием (FSM модель изменяется) √
  - \+ работает всегда
  - \+ возможное улучшение SDL модели в части случаев
  - – возможное ухудшение SDL модели в части случаев
  - – SDL модель хуже узнаваема
- Инкрементальный перенос изменений с MSC
  - – возможен в очень малом количестве случаев

5

Перейдем к генерации. Сразу отметим, что все существующие подходы осуществляют генерацию через введение промежуточной стадии – конечного автомата, который описывает поведение только одного объекта. Данный конечный автомат может как изменяться, так и не изменяться. В подходе без преобразования в полученной SDL модели лучше узнаваема MSC модель, в части случаев синтез не возможен из-за недостатков алгоритма, получаемая SDL модель иногда слишком громоздка.

С преобразованием. Этот подход работает всегда, он может как ухудшать, так и улучшать SDL модель, в SDL модели может хуже узнаваться исходная MSC модель.

Ставилась задача по инкрементальному переносу изменений, но она была решена лишь в очень малом количестве случаев.

Из всех подходов следует выбирать подход с преобразованием из-за более широкой его области применения.

Генерация: причины появления проблем в подходе с преобразованием

Обобщенная модель генерации:
MSC → FSM → SDL

Детализация преобразований:
MSC → FSM → det_FSM → min_FSM → SDL

Проблемы вносимые связкой det&min

- **неуправляемость процесса - получение SDL моделей не удобных для разработчика**
- **возможные неоправданные усложнения SDL**

6

Разберем причины появления проблем в подходе с преобразованием. Для этого детализируем его. Детерминизация автомата нужна для расширения области использования, а минимизация – поскольку полученный автомат может получиться слишком большим.

Появляющиеся проблемы из-за связки детерминизация – минимизация это:

• получение SDL моделей, не удобных для разработчика

• неоправданные усложнения генерируемого SDL

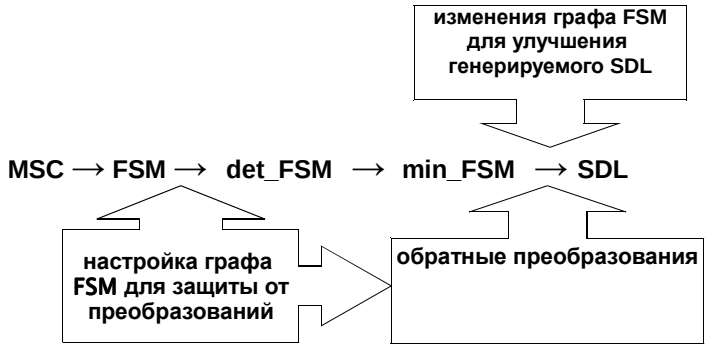
Дополнение:

1. Выбирается модель, которая позволяет осуществить синтез в любой ситуации
2. Рассказывается о преобразованиях автомата до генерации SDL
   - детерминизация нужна для реализации возможности синтеза
   - минимизация – поскольку автомат может получиться слишком громоздким
3. Поясняется, к каким проблемам это приводит

То есть, с одной стороны выбирается подход, при котором генерацию можно осуществить всегда, но за это приходится платить тем, что в середине процесса конечный автомат изменяется, что приводит к изменениям и в SDL. Иногда нежелательным.

Генерация: предлагаемые решения

Влияния на процесс генерации:



Полученные результаты:

- ✔ выделение SDL процедур
- ✔ защита от эффектов ухудшения SDL
- ✔ генерация SDL кода в виде, удобном для разработчика

7

Для того, чтобы избавиться от данных проблем, в процесс генерации предлагается вмешиваться. Мы либо вмешиваемся однократно перед генерацией из конечного автомата SDL, либо парно – до и после алгоритмов детерминизации и минимизации. При однократном вмешательстве мы работаем с графом, являющимся представлением конечного автомата. При парном – вносим изменения, которые защищают автомат от изменений, вносимых детерминизацией и минимизацией.

При этом мы получаем следующие результаты:

- выделение SDL процедур
- защита от эффектов ухудшения SDL
- генерация SDL кода в виде, удобном для разработчика

отсутствующие в других методах.

Дополнение:

Предлагается “вмешиваться” в автоматический алгоритм генерации SDL, после чего он становится автоматизированным, и это дает возможность получить следующие результаты.

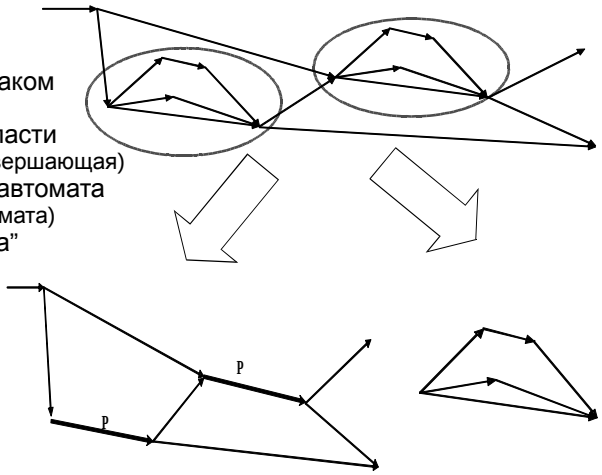
1. Дается схема внесения изменений в алгоритм генерации SDL
2. Описываются полученные результаты, которые дают дополнительные этапы внесения изменений

Следует учитывать, что данные изменения производятся на FSM, но результат их применения переносится в предметную область SDL.

Генерация: выделение SDL процедур

Выделяются области:

- Граф области является гамаком (один вход и один выход)
- Равны языки автоматов области (вход – начальная, выход – завершающая)
- В области нет завершения автомата (с точки зрения исходного автомата)
- Область “достаточно велика”



Также данный способ применим для выделения не повторяющихся (одиночных) процедур для улучшения структуры генерируемого SDL.

8

Для того, чтобы продемонстрировать что подразумевается под изменениями, вносимыми в граф конечного автомата, покажем преобразования, проводимые для выделения SDL процедур. Во всем графе ищутся области, которые в теории оптимизаций называются гамаком. Такие области имеют один вход и один выход. Должны быть равны языки, порождаемые данными областями. Все области заменяются на специальным образом маркированный переход, затем осуществляется генерация по графу с внесенными изменениями и по конечному автомату выделенной области.

Верификация: существующие подходы и их проблемы

Подходы:

1. MSC описывает трассы, которые «накладываются» на SDL модель
2. Состояния протокола - параллельное исполнение SDL и MSC моделей с ограничением потенциальной бесконечности

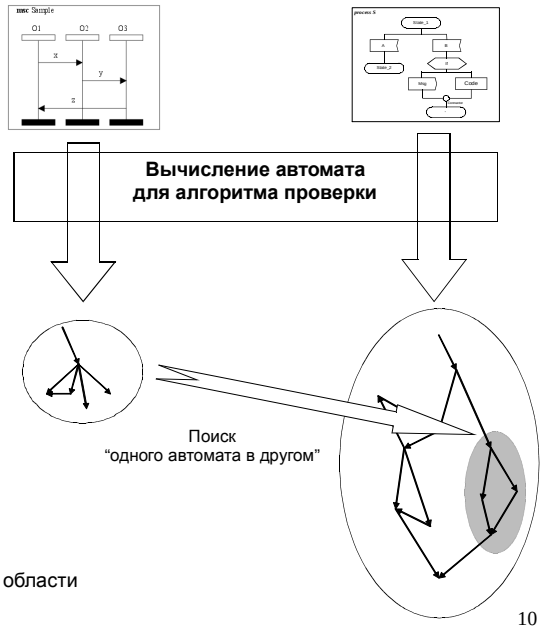
Проблемы, вызванные расхождением SDL и MSC моделей, допустимые с точки зрения человеческой логики:

- Реализация на SDL модели нескольких MSC ролей
- Разбиение MSC роли на несколько сервисов
- SDL модель циклична, а на MSC сценарии цикличности нет
- Введение дополнительных сообщений на SDL модели
- Разрозненность MSC сценариев
- Отсутствие на MSC диаграммах символа завершения

Теперь переходим к верификации. Основных подходов по верификации два. Это “наложение” MSC трасс на SDL модель и параллельное исполнение SDL и MSC моделей с ограничением потенциальной бесконечности. Однако при реальном использовании MSC и SDL модели отличаются, список некоторых возможных отличий приведен на слайде. Данные отличия допустимы с точки зрения человеческой логики, однако они вызывают проблемы при использовании данных подходов по верификации.

Верификация: предлагаемый подход

1. для каждого из объектов переход от MSC и SDL диаграмм к конечным автоматам с учетом предметной области
2. модификация автоматов:
   - разделение сообщений на “рассматриваемые” и “не рассматриваемые”
   - расширение множества завершающих состояний
   - “дублирование” автоматов для эмулирования работы очереди (уничтожение сообщений и конструкция Save)
3. поиск такого состояния на модифицированном SDL-автомате, что начиная с него можно “уложить” все возможные трассы, задаваемые модифицированным MSC-автоматом (трасс может быть бесконечное число из-за наличия циклов)
4. интерпретация результата в рамках предметной области



10

На этом слайде приведена схема алгоритма, решающего вышеописанные проблемы. Для SDL и MSC сначала осуществляется переход к конечным автоматам, затем данные автоматы преобразуются. После чего в измененном автомате SDL ищется такое состояние, что начиная с него будут порождаться все трассы, которые могут быть порождены модифицированным MSC автоматом. Потенциально таких трасс бесконечное число. Задача решается с помощью математики конечных автоматов, а потом решение интерпретируется в рамках предметной области.

Дополнение:

Дается идея подхода. Идея дается неконструктивно.

Сначала мы сводим предметную область к необходимым конечным автоматам, а затем осуществляем “поиск конечного автомата”. Эта задача сводится к включению одного языка в другой, что сводится к изоморфизму неких конечных автоматов. Это уже конструктивно.

Сложность:

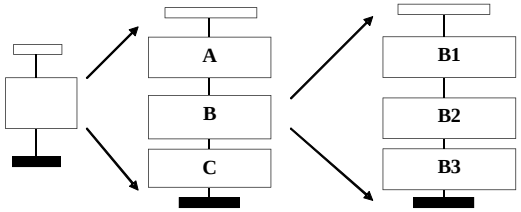
- детерминизация – экспоненциальная
- минимизация – n*ln(n)
- изоморфность – квадратичная

На самом деле из-за того, что автоматы берутся из MSC и SDL, экспонента не достигается и алгоритм выполняется в реальном времени.

Расширение описательных возможностей MSC: обоснование 1 (3)

Существующие возможности MSC.

Шаг 1: описание прямых веток, стандартная декомпозиция



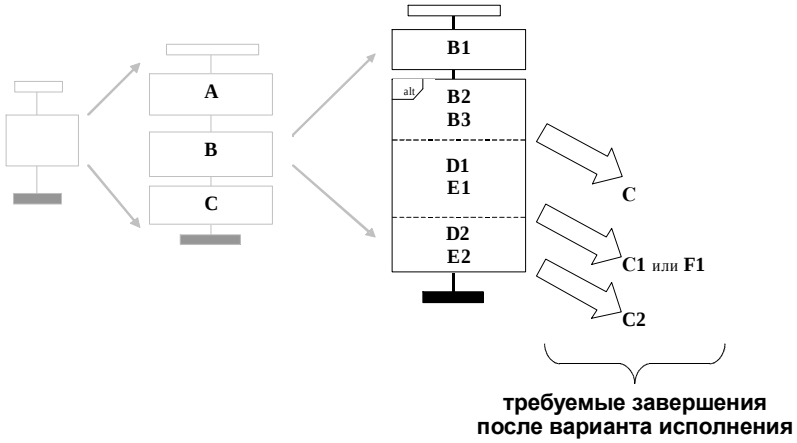
Для эффективного использования генерации и верификации MSC модель должна быть максимально более близкой (по полноте описания) к SDL модели. Рассмотрим проблемы существующего стандарта MSC при создании подобной модели.

Далее про расширение описательных возможностей MSC. Как уже упоминалось, для успешной работы генерации и верификации MSC модель должна быть достаточно информативной. Однако существующий стандарт MSC недостаточен для создания подобных моделей. Продемонстрируем это на примере. Для начала мы прорабатываем прямые ветки и проводим постепенную детализацию. Например, изначальный блок мы детализировали как три, затем блок B поделили еще на три.

Расширение описательных возможностей MSC: обоснование 2 (3)

Существующие возможности MSC.

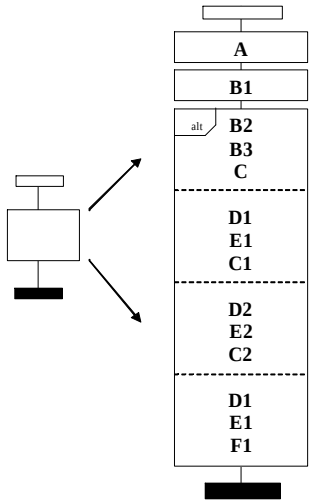
Шаг 2: проработка вариантов



12

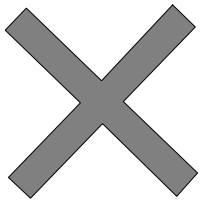
Вторым шагом мы начали добавлять в систему ошибочные ситуации, и обнаружили, что существуют другие варианты исполнения кроме B1-B2-B3. После того, как мы их нарисовали, мы обнаружили, что каждый их них требует своего варианта завершения, что изображено сбоку от сценария.

Расширение описательных возможностей MSC: обоснование 3 (3)



Существующие возможности MSC.

Шаг 3: описание вариантов на MSC



Отрицательные моменты:

1. *вернулись на детализируемую диаграмму, а детализирующую выкинули;*
2. *большое количество “перерисовок”;*
3. *полученная диаграмма громоздка.*

13

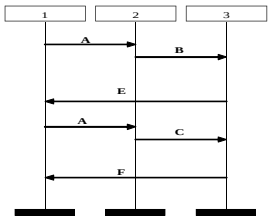
Для того, чтобы изобразить это в рамках существующего стандарта MSC нам пришлось отказаться от уровня детализации и перенести все на детализирующий уровень. Видно, что полученная диаграмма получилась слишком громоздкой, а так же применение данного метода требует большое количество “перерисовок”, что отнимает время.

Если проводить аналогии с АЯВУ, то у нас есть возможность вызова процедуры, но нет возможности описать функцию. Однако в реальных задачах практически всегда приходится закладываться на то, что в детализирующем сценарии произойдет ошибка, и, в зависимости от этого, придется менять поведение детализируемого сценария.

Расширение описательных возможностей MSC: идеи

Предлагаемая модель – это некоторое статическое описание, которое отражает логику взаимодействия объектов, и для каждого из объектов может быть превращено в конечный автомат.

Графические диаграммы, описывающие взаимодействия объектов



Текстовые описания, задающие логику стыковки сценариев

```
while function_1=2 do begin
      if function_2=0 then
            return 1
      else
            procedure_2
      fi;
      procedure_7;
end;
```

Как графические, так и текстовые описания делятся на два типа:
• “процедуры”
(не возвращают результат; соответствуют некоторому (сложному) сценарию);
• “функции”
(возвращают результат; разбивают множество вариантов поведения внутри на нумерованные классы).

14

Соответствующее решение было предложено. Оно является смесью графических MSC диаграмм согласно стандарту и текстовых вставок, описывающих логику стыковки сценариев. Как графические, так и текстовые описания делятся на “процедуры” и “функции”. “Процедуры” соответствуют обычному MSC сценарию, а “функции” являются набором сценариев, причем каждому из них сопоставлен номер. В зависимости от данного номера этот набор сценариев можно разобрать на необходимые группы.

Дополнение:

Про предлагаемое решение:

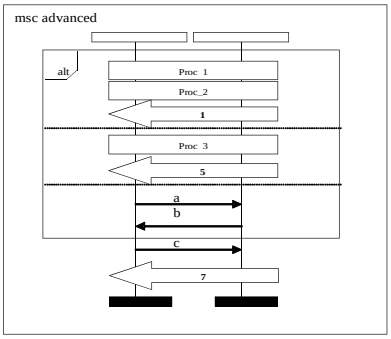
0. Подход был реализован в предложенном расширении MSC.
1. Рассказывается про описание расширение стандарта MSC:
   - графические диаграммы для описания взаимодействий (сохранены)
   - текстовые описания для описания логики стыковки сценариев
2. Интерпретация данной модели – на самом деле все конструкции **if, while ...** являются лишь способом представления, а на самом деле это лишь вариант записи конечного автомата.
3. Была разработана математическая модель, которая позволяет построить по данному расширению MSC конечный автомат.
4. Полученное расширение MSC позволяет более гибко описывать логику стыковки сценариев и, в частности, обратные ветви
5. Это дает возможность создавать более полные описания на расширенном MSC

Расширение описательных возможностей MSC: предложенное решение

Предлагаемые текстовые конструкции:
- `if then else fi`
- `case`
- `for (n)`
  повторить n раз
- `for ()`
  повторить 0..∞ раз
- `while`
- `alt`
- `opt`

С Pascal'е-подобным синтаксисом.

Возможное расширение графических возможностей MSC



В предложенном расширении MSC могут быть использованы конструкции, выписанные слева. Точно так же возможностью возвращать результат могут быть дополнены графические диаграммы. На диаграммы добавляется стрелка с номером, соответствующим возвращаемому результату.

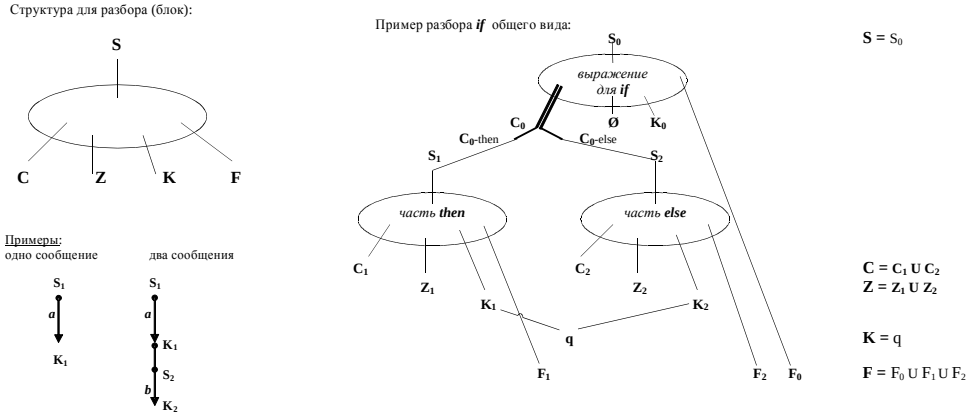
Дополнение:

Операторы **alt** и **opt** введены по аналогии с стандартом MSC. Построение С-подобных конструкций:

- **switch(break);**
- **for(continue,break);**
- **while(continue,break)**

также возможно. При необходимости строится и конструкция **goto.**

Расширение описательных возможностей MSC: структуры для разбора



Блок – конечно-автоматная структура для разбора, имеющая выделенные подмножества вершин:

- *вершина S обозначает начало блока;*
- *вершина K обозначает конец блока;*
- *множество вершин F обозначает завершение исполнения;*
- *Z – множество вершин выхода из "процедуры" ;*
- *C — множество вершин выхода из "функции", маркированных кодом возврата.*

16

Для разбора предложенного расширения MSC используется конечно-автоматная структура под названием “блок”. Она служит для разбора как графических, так и текстовых описаний.

- S обозначает начало блока;
- множество вершин F обозначает завершение исполнения;
- вершина K обозначает конец блока;

Ниже приведены блоки для одного сообщения, и для двух последовательно идущих сообщений. Для описания нескольких последовательно выполняющихся блоков S соединяется с K так, как показано на диаграмме для двух последовательных сообщений. C и Z служат для реализации механизма процедур и функций.

При разборе сценария, функции или процедуры – мы получаем блок. При разборе оператора на основе имеющихся блоков мы снова строим блок. В конце разбора мы также получаем блок, который сводим к обычному конечному автомату. Существуют процедуры проверки корректности при разборе.

Рядом приведен пример, демонстрирующий разбор оператора if и соответствующими операциями с блоками.

Дополнение:

Рассмотрим пример разбора оператора If у которого есть ветви then и else. Для того, чтобы его использовать, у нас должна присутствовать функция. На основе оператора, используемого в выражении if'а, мы разбиваем множество возвращаемых значений на две группы. Это же и дает нам разбиение сценариев. К одной группе мы “привешиваем” ветвь then, а к другой – ветвь else. Оператор if должен завершаться как после завершения ветви then, так и после завершения ветви else. Для этого мы вводим еще одну вершину q с добавлением соответствующих дуг на нее. Таким образом, при разборе оператора If мы снова получили блок.

Расширение описательных возможностей MSC: сравнение с существующими решениями

• Ряд решений (HMSC, UML Sequence и Collaboration, Use Case Maps…)
так же обладают недостаточными возможностями в части описания обратных ветвей

• Текстовые языки, нацеленные на описание взаимодействия
различных объектов (Pascal-FC, Occam, Ada, Java, Lotos…).
Они предназначены для описания поведения только одного объекта, а не группы.
Поэтому они проигрывают в выразительности

Похожие решения:

• LSCs (Life Sequence Charts)
  - она разбивает все сценарии на несколько (3) классов по критичности завершения, а в нашем случае вариантов поведения может быть произвольное количество
  - проигрывает по описательным возможностям
  - требует специальных средств для исполнения, а в нашем случае мы переходим в SDL

• CTP является комбинацией сетей Петри и MSC
предлагаемое решение выигрывает из-за использования АЯВУ против сетей Петри,
из-за более широкого использования АЯВУ на практике

17

Предлагаемое решение выигрывает у других графических диаграмм, поскольку логику стыковки сценариев удобнее описывать с помощью текста. Если сравнивать его с чисто текстовыми описаниями, то и здесь есть значительная разница, поскольку существующие текстовые описания предназначены для создания модели поведения только одного объекта, а на предложенном расширении описываются несколько, что дает выигрыш в восприятии системы.

Существует два подобных решения, LSCs и CTP. У первого мы выигрываем из-за возможности возвращать произвольное количество результатов и из-за того, что для его исполнения для него требуется отдельная модель, а в нашем случае мы переходим в стандартный SDL с возможностью использовать существующие средства.

У CTP предложенное решение выигрывает за счет того, что современному разработчику привычнее думать в терминах АЯВУ, нежели в терминах сетях Петри. Поскольку АЯВУ более широко распространены на практике.

Результаты

- ✓ Впервые поставлена задача настройки генерируемого SDL кода для повышения его читабельности и соответствия решаемой задаче.
- ✓ Разработан оригинальный алгоритм генерации SDL по MSC, решающий данную задачу.
- ✓ Разработана новая математическая модель верификации SDL по MSC.
- ✓ Разработано новое расширение MSC диаграмм, позволяющее создавать описания реальных систем.
- ✓ На базе данных разработок представлен оригинальный взгляд на совместное использование MSC и SDL моделей.
- ✓ Данные решения были проинтегрированы на базе существующей технологии REAL.
- ✓ Апробация разработанных решений на промышленных задачах.

Доклад окончен, благодарю за внимание!