Российская академия наук
Сибирское отделение
Институт систем информатики
имени А. П. Ершова

Э. Г. Тумуров

ТЕХНОЛОГИЯ СПЕЦИФИКАЦИИ КОММУНИКАЦИОННЫХ ПРОТОКОЛОВ

Препринт
146

Новосибирск 2007

В настоящей работе описывается технология спецификации коммуникационных протоколов в целях их реализации. Рассматривается набор коммуникационных протоколов с учетом характера взаимодействия, которое определяется комбинацией следующих параметров: блокированные или неблокированные прием/передача, надежная или ненадежная связь. Предлагаемая статья может рассматриваться как введение в проблематику коммуникационных протоколов.

Описываются разные схемы взаимодействия и организации передачи данных. Определяется набор структурных компонентов (узлы, каналы, сообщения, входные и выходные потоки, события, таймеры и др.), из которых составляется произвольный коммуникационный протокол. Описываются механизмы обеспечения надежной передачи по ненадежным каналам, такие как повторная передача, контрольное суммирование, подтверждения о приеме и др. В дополнение к спецификации протокола формулируется ряд свойств, отражающих корректность, безопасность и живость протокола. Эти свойства являются предметом верификации и записываются с использованием языка темпоральной логики.

Работа выполнена при поддержке РФФИ, грант № 04-01-00272.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

E. G. Tumurov

THE TECHNOLOGY OF SPECIFICATION OF COMMUNICATION PROTOCOLS

Preprint
146

Novosibirsk 2007

The work presents the technology of communication protocol specification. Several communication protocols are considered for the following aspects of communication: blocked or non-blocked receiving/sending, reliable or unreliable communication. This paper can be considered as an introduction to problems of communication protocols.

Some communication and data transfer patterns are described. The following communication protocol components are defined: nodes, channels, messages, input and output streams, events, timers, etc. Techniques of reliable transfer over unreliable channels such as retransmission, check summing, reception of acknowledgment, etc. are described. Additional properties of protocol specifications set are given. They are correctness, safety and liveness properties. They are usually expressed in some language of temporal logic.

1. ВВЕДЕНИЕ

Коммуникационный протокол — стандарт, определяющий правила передачи информации между двумя устройствами, называемыми *узлами*. Правила передачи определяют представление данных, способ аутентификации (процедуру проверки подлинности объекта взаимодействия) и механизм обеспечения надежности передачи. Данные передаются узлом-*отправителем* и принимаются узлом-*получателем*. Узлы взаимодействуют через устройства, называемые *каналами*. Примеры узлов: телефонные станции, компьютеры в сети Интернет и др.

Информация по каналу передается в виде сообщений. Каждое сообщение включает в себя передаваемые данные, адрес назначения сообщения и другую информацию.

Каналы потенциально ненадежны. Возможны потеря, порча, дублирование, изменение порядка доставки сообщений. Передача данных реализуется по физическим линиям связи через физические устройства, которые при высокой надежности передачи тем не менее могут давать ошибки.

Обнаружение ошибок передачи реализуется включением в сообщение избыточной информации. Обычно для этой цели используются *контрольная сумма* сообщения, посчитанная по специальному алгоритму, биты четности и др.

Передача сообщений подразделяется на *синхронную* и *асинхронную*. С начала синхронной передачи передающая сторона приостанавливает свою работу (блокируется) до тех пор, пока сообщение не будет принято адресатом или будет получен отказ. При асинхронной передаче передающая сторона отправляет сообщение по каналу и продолжает работу, не дожидаясь завершения передачи.

Наше рассмотрение коммуникационных протоколов ограничивается собственно передачей данных. Адресация, аутентификация, соединения и другие аспекты протоколов не рассматриваются.

Протоколы относятся к классу программ с процессной спецификацией [1,2]. Спецификация называется *процессной*, если эффект исполнения программы описывается в виде процесса, функционирующего одновременно с другими процессами и обменивающегося с ними информацией. Процессная спецификация сложнее предикатной [1], которая может рассматриваться как частный случай процессной.

Существуют следующие модели описания процессов: машина конечных состояний [3], сеть Петри [4], машина абстрактных состояний [5] и др. В дополнении к этой классификации следует рассматривать следующие модели взаимодействия параллельных процессов: модель CSP [6] Т. Хоара, модель CCS [7] Р. Милнера, π-исчисление [8], join-исчисление [9].

Для разработки программ с процессной спецификацией используются языки и системы программирования, предоставляющие средства описания процессов и их взаимодействия с окружением. Это языки спецификаций SDL [10], UML [11], AsmL [12] и другие. Для построения программ, работающих в реальном времени, разработана объектно-ориентированная методология Real [13], базирующаяся на языках спецификаций UML, SDL и интегрирующая в себе также ряд других методологий разработки информационных систем. Известен также язык REAL [14], комбинирующий спецификации типа SDL с описанием свойств процессов на языке пропозициональной динамической логики.

Язык процессной спецификации состоит из ядра и собственно средств описания процессов. В перечисленных и многих других языках в качестве ядра используется некоторый язык императивного программирования. Исключениями являются языки Erlang [15, 16] и JoCaml [17]. Их ядром является язык функционального программирования. Язык Erlang является расширением языка типа Lisp и используется для реализации телекоммуникационных систем. Язык JoCaml построен на базе объектно-ориентированного языка OCaml [18] из семейства ML и представляет реализацию модели join-исчисления [9] взаимодействия процессов.

В данной работе представлена технология спецификации коммуникационных протоколов. Структура спецификации представлена в начале разд. 3. Моделью спецификации является машина конечных состояний в виде гиперграфа состояний [1]. При описании процесса, реализуемого программой, осуществляется иерархическая декомпозиция процесса. Каждая часть процесса специфицируется независимо. Гиперграфовая модель дает возможность произвольной и гибкой декомпозиции алгоритма, что определяет существенное преимущество по сравнению с моделью, используемой в языке SDL [10]. В частности, любой алгоритм можно декомпозировать введением всего лишь одного внутреннего состояния. Взаимодействие параллельных процессов специфицируется с использованием механизма сообщений. Поведение параллельных процессов можно также представить в виде графа метасостояний [19, 20].

В работе сначала описывается общая модель коммуникационного протокола. Представлены разные схемы взаимодействия и организации пере-

дачи данных. Определяется набор структурных компонентов коммуникационных протоколов. Описываются дефекты каналов связи и механизмы обеспечения надежной передачи по ненадежным каналам, такие как повторная передача, контрольное суммирование, подтверждения о приеме и др. Приводится ряд примеров спецификации протоколов с различными свойствами каналов. Формулируется ряд свойств, отражающих корректность, безопасность и живость протокола. Эти свойства записываются с использованием языка темпоральной логики.

2. МОДЕЛЬ ПРОТОКОЛА ПЕРЕДАЧИ ДАННЫХ

Передача данных по каналам от отправителя к получателю осуществляется частями, называемых *блоками*. Отправитель считывает блок из *входного потока* и оформляет его в виде сообщения для посылки по каналу. Получатель, получив сообщение с блоком, записывает его в *выходной поток*. Во время работы в выходной поток попадают блоки, считанные из входного потока в порядке их считывания. Канал, в котором передача сообщений идет от отправителя к получателю, будем называть *каналом данных*, а канал, в котором передача идет в обратном направлении — *каналом ответов*.

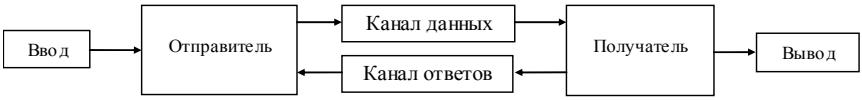

Рис.1. Взаимодействие отправителя и получателя

Спецификация протокола записывается в виде программы на языке P [21], содержащем средства описания процессов, взаимодействующих с помощью сообщений.

Определен некоторый тип блока данных Блок, функция ввод для чтения блока из входного потока и функция вывод для записи блока в выходной поток.

```
type Блок
process ввод( : Блок b)
process вывод(Блок b : )
```

Отправитель и получатель моделируются процессами Отправитель и Получатель.

process Отправитель();
process Получатель();

Сообщение идентифицируется именем и набором параметров, содержащихся в сообщении.

message <имя> (<список типов параметров>)

Оператор отправки сообщения другому процессу:

<процесс-приемник> **!** <имя-сообщения>(<список выражений >)

Оператор приема сообщения:

receive <имя>(<список переменных>): <оператор>;

Допустим, что процесс, исполняющий оператор **receive,** получил сообщение с именем <имя>. Исполнение оператора заключается в присваивании перечисленным переменным соответствующих значений, поставляемых сообщением, и дальнейшим исполнением оператора <оператор>. Если процесс не получил сообщения с указанным именем, то исполнение оператора блокируется до поступления требуемого сообщения.

Также для приема сообщений используется оператор общего вида:

receive <имя1>(<список переменных1>): <оператор1> **or** ... **or**
<имяN>(<список переменныхN>): <операторN>
after <выражение>: <оператор>
end

Оператор ждет получения одного из сообщений с именами <имя1>, … или <имяN>. При поступлении сообщения с именем <имяK> параметры этого сообщения присваиваются переменным из <списка переменныхK> и исполняется <операторK>. Этот оператор также является блокирующим. Конструкция <выражение> типа **nat** определяет значение времени в миллисекундах, по истечении которого срабатывает <оператор>, указанный завершителем **after,** и процесс продолжает работу. Конструкция **after** <выражение>: <оператор> может быть не указана, тогда процесс блокируется до поступления нужного сообщения.

У каждого процесса есть *очередь* сообщений. Эта очередь содержит те сообщения, которые были получены из среды. Когда процесс выполняет

оператор приёма сообщения **receive**, в очереди ищется первое подходящее сообщение. Если оно было найдено, то это сообщение убирается из очереди. Если подходящих сообщений в очереди не оказалось, то процесс блокируется, пока не придёт подходящее сообщение, либо не исчерпается *максимальное время ожидания (тайм-аут)*. При отправке сообщения процесс не блокируется.

2.1. Обеспечение надежности передачи сообщений

При передаче через канал возможны следующие дефекты:

- изменение порядка — сообщение, посланное позже другого, приходит раньше;
- потеря — посланное сообщение никогда не будет получено;
- порча — посланное сообщение доходит до получателя с ошибками;
- дублирование — некоторые сообщения дублируются один или несколько раз.

Для обеспечения надежности передачи по каналу с дефектами применяют следующие способы:

Порча. *Первый способ.* С каждым посылаемым блоком передается его контрольная сумма. Получатель, получив сообщение, может распознать, был ли испорчен блок, и запросить повторную посылку.

Второй способ. Кодирование данных с учетом характера изменения сообщений каналами. Существуют коды, обнаруживающие ошибки, и коды, исправляющие ошибки.

Потеря. Блоки нумеруются в порядке их считывания из входного потока. Получатель записывает блоки в порядке нумерации (для протокола АВР используется чередование 0 и 1). Получатель на каждый принятый блок отправляет подтверждение о приеме с номером принятого блока. Если отправитель не получает подтверждение в течение некоторого времени, он посылает его повторно.

Изменение порядка, **дублирование**. Блоки нумеруются в порядке их считывания из входного потока. Получатель записывает блоки в порядке нумерации. Из полученных блоков с одинаковыми номерами в выходной поток записывается только один.

В следующем разделе эти способы будут продемонстрированы на примере протоколов, работающих с различными каналами.

3. СПЕЦИФИКАЦИЯ ПРОТОКОЛОВ

Спецификация программы реального времени включает: спецификацию окружения, спецификацию декомпозиции системы, иерархическую спецификацию каждого процесса системы и спецификацию временных свойств системы.

В настоящей статье рассматривается эволюция коммуникационных протоколов от самых простых и неэффективных до современных эффективных протоколов, реально используемых на практике и соответствующих определенному набору требований.

Спецификация окружения всех определяемых ниже протоколов передачи данных включает процессы ввод и вывод, тип Блок:

```
type Блок
process ввод( : Блок b)
process вывод(Блок b : )
```

Спецификация декомпозиции определяет процессы Отправитель и Получатель, сообщения, посредством которых взаимодействуют процессы, и общие структуры данных.

3.1. Простейший коммуникационный протокол

Рассмотрим коммуникационный протокол с синхронным и надежным каналом данных. Канал ответов не используется. По каналу данных передаются сообщения с блоками данных.

Процессы отправителя и получателя выполняются параллельно. В программу дополнительно вводится сообщение OK для синхронизации процессов. В каждый момент времени между этими процессами передается не более одного сообщения.

```
// протокол определяется как параллельная работа процессов
process Протокол1() { Отправитель() || Получатель() }
message ДАННЫЕ(Блок блок), OK();       //определение сообщений
process Отправитель() {
  loop: Блок блок = ввод()         //считывание блока из входного потока
        Получатель ! ДАННЫЕ(блок);//передача сообщения с блоком данных
        receive OK();                    //прием подтверждения
        #loop                            //переход в начало цикла
}
```

```
process Получатель() {
  loop: receive ДАННЫЕ(Блок блок); //получение сообщения
                                   //с блоком данных
        Отправитель ! ОК()         //отправка подтверждения о приеме
        вывод(блок)                //запись блока в выходной поток
        #loop                      //переход в начало цикла
}
```

3.2. Протокол передачи по каналу с возможной порчей данных

При передаче возможна порча блока данных сообщения. Обнаружение ошибок достигается проверкой контрольной суммы блока. С каждым блоком данных передается контрольная сумма, посчитанная функцией контрольнаяСумма. При приеме сообщения вычисляется контрольная сумма полученного блока и сравнивается с суммой, хранящейся в сообщении. Мы исключаем из рассмотрения случай, когда блок испорчен, а контрольная сумма совпадает. Отметим, что в данном протоколе не допускается потеря и дублирование сообщений.

Посылка сообщений процессом Отправитель происходит с помощью процесса послатьДанные, который посылает сообщения ДАННЫЕ с возможной порчей блока при передаче процессу Получатель. Используется внешняя функция возможноИспортить, которая недетерминировано портит или сохраняет блок данных.

Дополнительных сообщений для синхронизации не требуется.

```
process Протокол2() { Отправитель() || Получатель() }
message ДАННЫЕ(Блок блок, int сумма), ОШИБКА(), ОК()
predicate контрольнаяСумма(Блок блок : int сумма)

process Отправитель() {
  new:  Блок блок = ввод()
  send: послатьДанные(блок, контрольнаяСумма(блок):)
        receive ОШИБКА():    #send    //передать блок ещё раз
        or ОК():             #new     //передать новый блок
        end
}
```

```
process Получатель() {
 loop:  receive ДАННЫЕ(Блок блок, int сумма);
        if контрольнаяСумма(блок) <> сумма then
              Отправитель ! ОШИБКА();
        else
              Отправитель ! ОК();
              вывод(блок)
        end
        #loop
}
process возможноИспортить(Блок блок : Блок блок');
                                  //возможная порча содержимого блока
process послатьДанные(Блок блок, int сумма : ) {
        Получатель ! ДАННЫЕ(возможноИспортить(блок), сумма);
}
```

3.3. Протокол n-ABP

n-ABP — модификация протокола чередования битов (alternating bit protocol) [19]. Все блоки данных нумеруются последовательно натуральными числами. Номер блока данных передается при каждой передаче. Отправитель, получив блок данных, отправляет ответ с номером принятого блока. Если отправитель не получает ответ (с номером посланного блока) в течение определенного времени, то он делает повторную посылку блока данных. Возможны порча, потеря, изменение порядка и дублирование сообщений. Для контроля ошибок используются контрольные суммы.

Посылка сообщений протокола моделируется внешними процессами послатьДанные и послатьОтвет. Процесс послатьДанные посылает сообщение ДАННЫЕ процессу Получатель, а процесс послатьОтвет посылает сообщение ОК процессу Отправитель. Процесс послатьДанные моделирует дефекты канала данных: он может потерять, испортить, продублировать сообщение или поменять порядок сообщений. Процесс послатьОтвет моделирует дефекты канала ответов: он может потерять, продублировать сообщение или поменять порядок сообщений.

```
process nABP() { Отправитель() || Получатель() }
message ДАННЫЕ(Блок блок, int сумма, int номер), ОК(int номер);
predicate посчитатьСумму(Блок блок : int сумма);
nat timeout; //время ожидания в миллисекундах
```

```
process Отправитель() {
        int прочитано = 0;                  //номер текущего блока
 read: Блок блок = ввод()
        прочитано = прочитано + 1
 send: послатьДанные(блок, посчитатьСумму(блок), прочитано)
        receive ОК(int номер):
                if номер <> прочитано then #send
                else #read
                end
        after timeout: #send
        end
}
process Получатель() {
        int записано = 0;
 loop: receive ДАННЫЕ(Блок блок, int сумма, int номер):
                if посчитатьСумму(блок) <> сумма then      #loop
                else
                        послатьОтвет(номер)
                        if номер <> записано+1 then      #loop
                                                  //проверка номера блока
                        else #write
                        end
                end
        end
 write: вывод(блок);
        записано = записано + 1;
        #loop
}
process послатьДанные(Блок блок, int сумма, int номер);
process послатьОтвет(int номер);
```

3.4. Протокол скользящего окна

Протокол скользящего окна — эффективный протокол передачи данных. При передаче блоки посылаются друг за другом без ожидания подтверждения. При этом блоки сохраняются в буфере — «окне». Новые неподтвержденные блоки добавляются в конец окна, а подтвержденные блоки удаляются из начала окна — так происходит «скольжение» окна.

3.4.1. Спецификация окружения

```
type Блок
process ввод( : Блок b)
process вывод(Блок b : )
```

3.4.2. Спецификация декомпозиции

Размер окна ограничен. В начале каждой итерации, если можно увеличить окно, то Отправитель считывает очередной блок для передачи, передает его и сохраняет в окно. При этом для каждого посланного сообщения устанавливается таймер с номером блока. Когда срабатывает таймер, Отправитель делает повторную посылку блока. Повторная посылка также происходит при получении сообщения ОШИБКА от Получателя.

```
process ПротоколСкользящегоОкна() { Отправитель() || Получатель() }

type Номер = int;
int WMAX = 10; //макс. возможный размер окна

message ТАЙМЕР(Номер номер);
message ОК(Номер номер);
message ОШИБКА(Номер номер);
message ДАННЫЕ(Номер номер, Блок блок, int сумма);

//    внешний процесс: сначала останавливает таймер,
//    если таймер для номера уже есть, во время срабатывания таймера,
//    то Отправителю приходит сообщение ТАЙМЕР
process запуститьТаймер(Номер номер);
process остановитьТаймер(Номер номер);
//    подсчёт контрольной суммы
predicate посчитатьСумму(Блок блок : int сумма);
//    посылает сообщение ДАННЫЕ
process послатьДанные(Номер номер, Блок блок, int сумма);
type ТипОтвета = enum {ОШИБКА_Т, ОК_Т};
//    посылает ответ ОК или ОШИБКА
process послатьОтвет(Номер номер, ТипОтвета тип);

process Отправитель();
process Получатель();
```

3.4.3. Спецификация процессов

Когда Отправитель получает подтверждение ОК(номерБлока), то это означает, что все сообщения из окна с номером меньше номерБлока подтверждены. Отправитель останавливает таймеры переданных блоков и убирает блоки из окна.

```
//вспомогательные предикаты
predicate вОкне(Номер н, Номер низ, Номер верх: bool результат) {
        результат = низ ≤ н and н ≤ верх
}
predicate размер(Номер низ, Номер верх : Номер результат) {
        результат = верх — низ + 1
}
predicate след(Номер н : Номер следНомер) { следНомер = н + 1 }
predicate пред(Номер н : Номер предНомер) { предНомер = н - 1 }

//подпроцесс  Отправителя
process передатьБлок(Номер номер, Блок блок) {
        послатьДанные(номер, блок, посчитатьСумму(блок));
        запуститьТаймер(номер);
}

process Отправитель() {
        Номер началоОкна = 0;          //нижняя граница окна
        Номер конецОкна = 0;           //верхняя граница окна
        Номер максОкно = WMAX;         //ограничение размера окна
        Блок буфер[WMAX];        //буфер
 send: if размер(началоОкна, конецОкна)<максОкно then
                конецОкна = след(конецОкна);
                буфер[конецОкна] = ввод(Блок блок);
                передатьБлок(конецОкна, буфер[конецОкна%WMAX]);
        end
 accept: if ОШИБКА(Номер номерБлока) then
                if вОкне(номерБлока, началоОкна, конецОкна) then
                        передатьБлок(номерБлока, буфер[номерБлока%WMAX])
                end
                #accept
        elsif ОК(Номер номерСледБлока) then
                while вОкне(пред(номерСледБлока),
                                началоОкна, конецОкна) do
                        остановитьТаймер(началоОкна);
```

```
                    началоОкна = след(началоОкна);
                end
                #accept
        end
  timeout: if ТАЙМЕР(номерБлока) then
                передатьБлок(номерБлока, буфер[номерБлока%WMAX]);
                #timeout
        end;
        #send
}
```

Получатель имеет буфер-окно полученных блоков и список номеров полученных блоков окна. Размер окна всегда максимален. Приняв сообщение ДАННЫЕ, Получатель сначала проверяет блок на ошибки с помощью проверки контрольной суммы. Если полученный блок испорчен, его номер в текущем окне, и блок с таким номером ещё не был получен, то посылается сообщение ОШИБКА с номером блока. Если блок неиспорчен, тогда он заносится в окно, если такой блок ещё не был получен ранее. Затем, если в начале окна есть последовательность полученных блоков, эти блоки записываются в выходной поток и удаляются из окна. После посылается подтверждение ОК с номером начала окна.

```
process Получатель() {
        Номер началоОкна = 0;          //окно входящих сообщений
        Номер конецОкна = WMAX-1;  //окно входящих сообщений
        Блок буфер[WMAX];              //буфер входящих сообщений
        array of 0.. WMAX-1 bool  получено;
                                        //флаги полученных сообщений
        forAll i = 0 .. WMAX-1 do получено[i] = false end
  loop: receive ДАННЫЕ(Номер номерБлока, Блок блок, nat суммаБлока):
                if посчитатьСумму(блок) <> суммаБлока then
                        if вОкне(номерБлока, началоОкна, конецОкна)
                          and ¬(получено[номерБлока%WMAX]) then
                                послатьОтвет(номерБлока, ОШИБКА_Т);
                        end
                else
                        if вОкне(номерБлока, началоОкна, конецОкна)
                          and ¬(получено[номерБлока%WMAX]) then
                                буфер[номерБлока%WMAX] = блок;
                                получено[номерБлока%WMAX] = true;
                                while (получено[началоОкна%WMAX]) do
                                        вывод(буфер[началоОкна%WMAX]);
```

```
                                получено[началоОкна%WMAX] = false;
                                началоОкна = след(началоОкна);
                                конецОкна = след(конецОкна);
                            end
                        end
                        послатьОтвет(началоОкна, ОК_Т);
                    end
                end
            }
```

4. СВОЙСТВА ПРОТОКОЛОВ

Корректная работа коммуникационного протокола может быть выражена в виде набора временных свойств. Эти свойства формализуются на языке *темпоральной логики* [22] и могут использоваться для верификации. Описание свойств использует модель глобального дискретного ветвящегося времени [23]. Язык темпоральной логики базируется на языке логики предикатов первого порядка.

4.1 Язык темпоральной логики

Набор значений всех переменных программы (для всех её процессов) в текущий момент исполнения назовём *состоянием памяти* программы $V=(v_1,v_2,...v_n)$. Кроме того, состояние памяти содержит место исполнения каждого активного процесса — для этой цели вводится специальная переменная, фиксирующая место исполнения в процессе.

Каждый процесс рассматривается как последовательность *атомарных действий*. Постулируется, что выполнение атомарного действия не зависит от параллельных действий в других процессах. Таким образом, мы можем считать, что при выполнении атомарного действия другие процессы приостановлены, т.е. разные атомарные действия реализуются в режиме чередования. Эта модель исполнения определяет семантику чередования (interleaving).

Свойства протоколов записываются на языке ветвящейся темпоральной логики CTL [23]. Временные формулы описывают свойства *деревьев вычислений*. Такое дерево образуется из начального состояния и всех возможных вариантов исполнения программы. Конкретное исполнение программы представляется в виде пути в этом дереве. Путь вычисления $\sigma=(s_0,s_1,...,s_k,...)$ состоит из последовательности состояний s_k памяти про-

граммы. Переходы между состояниями реализуются исполнениями атомарных действий, перемешанных из разных параллельных процессов.

Формулы ветвящегося времени используют *кванторы пути* и *темпоральные операторы*. Кванторы пути описывают структуру ветвления в дереве вычислений: квантор **A** означает истинность (соответствующей подформулы) для всех путей вычисления, а квантор **E** — для некоторого пути вычисления.

Основные темпоральные операторы:

- оператор следования ○ требует, чтобы свойство соблюдалось в следующем состоянии пути;
- оператор «когда-нибудь в будущем» ◊ указывает, что свойство будет соблюдаться в каком-то последующем состоянии пути;
- оператор «когда-то в прошлом» ♦ говорит, что свойство соблюдалось в каком-то предшествующем состоянии пути;
- оператор инвариантности □ показывает, что свойство соблюдается в текущем и каждом последующем состоянии пути;
- оператор условного ожидания «до тех пор, пока» U комбинирует два свойства: он выполняется, если на пути имеется состояние, в котором соблюдается второе свойство, и при этом каждое предшествующее на этом пути состояние обладает первым свойством.

Каждый темпоральный оператор должен следовать непосредственно за квантором пути.

Формулы пути строятся из темпоральных операторов над *формулами состояний*. Формулы состояния способны быть истинными на одном состоянии, а формулы пути — на протяжении некоторого пути. Формула состояния представляет собой формулу языка логики исчисления предикатов над элементами состояния памяти программы. *Формула пути* строится из *формул состояний*, логических операторов $\wedge$, $\vee$, $\neg$, и темпоральных операторов.

Каждое состояние памяти s_k определяет *интерпретацию* для формул состояния, т.е. по данным значениям переменных из s_k определяется, истинна формула или нет. Для некоторого пути вычисления $\sigma=(s_0,s_1,\ldots,s_k,\ldots)$ определим интерпретацию формулы пути p в момент времени j: $(\sigma,j) \models p$. Семантика формул темпоральной логики определяется следующими тождествами, где p и q — произвольные временные формулы:

- $(\sigma,j) \models p \quad \cong \quad s_j \models p$
- $(\sigma,j) \models \neg p \quad \cong \quad s_j \models \neg p$
- $(\sigma,j) \models p \vee q \cong \quad (\sigma,j) \models p$ *или* $(\sigma,j) \models q$

- $(\sigma,j) \models p \wedge q \cong (\sigma,j) \models p$ и $(\sigma,j) \models q$
- $(\sigma,j) \models p\,U\,q \cong \exists k, k>j: (\sigma,k) \models q$ и $\forall i, j \le i < k: (\sigma,i) \models p \wedge \neg q$
- $(\sigma,j) \models \circ p \cong (\sigma,j+1) \models p$
- $(\sigma,j) \models \Box p \cong \forall k, k \ge j: (\sigma,k) \models p$
- $(\sigma,j) \models \Diamond p \cong \exists k, k \ge j: (\sigma,k) \models p$
- $(\sigma,j) \models \blacklozenge p \cong \exists k, k \le j: (\sigma,k) \models p$

Временная формула получается из формул пути добавлением квантора пути **A** или **E** перед каждым темпоральным оператором. Таким образом, временная формула применяется над деревом вычислений.

Программа *имеет свойство* p, если временная формула p истинна над полным деревом вычисления программы.

Специально выделяют следующие классы временных свойств программ:

свойство *живости* — некоторое свойство, определяющее, что периодически должно происходить что-то хорошее;

свойство *безопасности* — некоторое свойство, определяющее, что не должно происходить что-то плохое;

свойство *корректности* —в процессе выполнения при определенных начальных условиях получается нужный результат.

4.2 Пример спецификации временных свойств

Рассмотрим методы спецификации свойств на примере протокола передачи данных по каналу с возможной порчей данных (см. 3.2). Для формулировки временных свойств программа инструментируется дополнительными переменными, операторами и метками:

```
process Отправитель() {
        int n = 0;
  new:  n = n + 1; Блок a = ввод()
  @read:
  send: послатьДанные(a, контрольнаяСумма(a):)
        receive ОШИБКА():  #send              //передать блок ещё раз
        or ОК():           #new               //передать новый блок
        end
}
process Получатель() {
        int m = 0;
  loop: receive ДАННЫЕ(Блок b, int сумма);
        if контрольнаяСумма(b) <> сумма then
```

```
                Отправитель ! ОШИБКА();
        else
@ok:            Отправитель ! ОК();
                m = m + 1; вывод(b)
@write:
        end
        #loop
}
```

Программу протокола инструментируем переменными n и m типа **int**. Перед операцией ввод n увеличивается на 1, перед операцией вывод m увеличивается на 1. n и m инициализируются 0. Также инструментируем программу метками: @ок, указывающую на операцию посылки ответа ОК Получателем, @read, указывающую на завершение операции ввод, и @write, указывающую на завершение операции вывод. Место исполнения процесса Отправитель обозначает переменная pcO, а место исполнения процесса Получатель — pcП. Переименовываем совпадающие имена локальных переменных процессов, чтобы они имели уникальные имена.

Таким образом, имеем V=(n, m, a, b, сумма, pcO, pcП).

Определим временные свойства протокола:

1) **A** □ ((**E** ◊ pcO = @read) ⇒ (**E** ◊ pcO = @write))
   если новые блоки данных считываются, то происходит запись блоков данных;
2) **A** □ ((pcП = @write ∧ m=k ∧ c=b) ⇒ **A**♦(pcO = @read ∧ n=k ∧ c=a))
   для произвольных k и c: если записан k-й блок, то этот блок был считан под номером k из входного потока.

Свойство 1 является свойством живости, а свойство 2 — свойством безопасности. Свойство 2 также является свойством корректности: выходной поток совпадает с началом входного потока.

5. ЗАКЛЮЧЕНИЕ

Данная работа может рассматриваться как введение в проблематику коммуникационных протоколов. Понятие коммуникационного протокола вводится сначала для простейших способов взаимодействия узлов. Далее описывается все более усложняющийся набор протоколов, учитывающих особенности каналов связи (порча, потеря, дублирование, изменение порядка) и требования к эффективности передачи. При описании протоколов

используется технология спецификации программ реального времени. Эта технология базируется на модели передачи данных, языке спецификаций P и языке ветвящейся темпоральной логики [23].

Используемая технология предоставляет средства для явного компактного и гибкого описания протоколов в целях их эффективной реализации.

СПИСОК ЛИТЕРАТУРЫ

1. Шелехов В.И. Анализ общего понятия программы. // Методы предикатного программирования. Вып.2 / ИСИ СО РАН. — Новосибирск, 2006. — С. 7–16.
2. Шелехов В.И. Иллюстрация процессной спецификации на примере программы гадания на кофейных зернах // Методы предикатного программирования. Вып.2. — Новосибирск, 2006. — С. 35–39.
3. Kain R.Y. Automata Theory: Machines and Languages. — McGraw-Hill, NY, 1972. — 301 p.
4. Питерсон Дж. Теория сетей Петри и моделирование систем. — М. Мир, 1984. — 264 с.
5. Gurevich Y. Sequential Abstract State Machines capture Sequential Algorithms // ACM Transactions on Computational Logic. — 2000. — Vol. 1, N. 1. — P. 77–111.
6. Hoare C.A.R. Communicating Sequential Processes. — Prentice-Hall, 1985.
7. Milner R. A Calculus of Communicating Systems // Lect. Notes Comput. Sci. — 1980. — Vol. 94.
8. Milner R. Communicating and Mobile Systems: the Pi-Calculus. — Cambridge University Press, 1999.
9. Fournet C., Gonthier G. The join-calculus model of concurrency: The reflexive chemical abstract machine and the join-calculus. — POPL'96, 1996.
10. ITU Recommendation Z.100: Specification and Description Language (SDL-96), ITU General Secretariat — Sales Section, Place de Nations, CH-1211 Geneva 20. — 218 p.
11. UML semantics, version 1.1. — 1997. — 162 p. — http://www.rational.com/uml
12. The AsmL webpage. — http://research.microsoft.com/foundations/AsmL/.
13. Терехов А.Н, Романовский К.Ю., Кознов Д.В. и др. Real: методология и CASE-средство для разработки систем реального времени и информационных систем // Программирование. — 1999. — № 5. — С. 44–52.
14. Непомнящий В.А., Шилов Н.В. REAL 92: комбинированный язык спецификаций для систем и свойств взаимодействующих процессов реального времени // Программирование. — 1993. — № 6. — С. 64–80.
15. B. Däcker. Concurrent Functional Programming for Telecommunications: A Case Study of Technology Introduction / Licenciate Thesis, Department of

Teleinformatics. — TRITA-IT AVH 00:08, ISSN 1403-5286, Royal Institute of Technology. — Stockholm, 2000.
16. http://www.erlang.org/
17. Fournet C., Fessant F., Maranget L., Schmitt A.. JoCaml: A Language for Concurrent Distributed and Mobile Programming // Advanced Functional Programming 2002. — Lect. Notes Comput. Sci. — 2002. — Vol. 2638. — P. 129–158. (LNCS)
18. Leroy X. et. al. The Objective CAML system 3.05. — http://caml.inria.fr.
19. Тумуров Э. Г. Спецификация и верификация протокола чередования битов // Методы предикатного программирования. Вып.2. — Новосибирск, 2006. — С. 64–74.
20. Шелехов В. И., Каличкин С.В. Спецификация, верификация и реализация системы измерения смещений и деформаций смежных конструкций // Методы предикатного программирования. Вып.2. — Новосибирск, 2006. — С. 82–104.
21. Шелехов В. И. Язык спецификации процессов // Методы предикатного программирования. Вып.2. — Новосибирск, 2006. — С. 17–34.
22. Pnueli A. The Temporal Logic of Programs. // Proc. of the 18th IEEE Symp. on Foundations of Computer Science (FOCS 1977). — 1977. — P. 46-57.
23. Clarke E. M., Grumberg O., Peled D. Model Checking. — The MIT Press, 2000.

Э. Г. Тумуров

ТЕХНОЛОГИЯ СПЕЦИФИКАЦИИ КОММУНИКАЦИОННЫХ ПРОТОКОЛОВ

Препринт
146

Рукопись поступила в редакцию 06.11.07
Рецензент Н. О. Гаранина
Редактор Т. М. Бульонкова

Подписано в печать 28.12.07
Формат бумаги 60 × 84 1/16 Объем 1.3 уч.-изд.л., 1.4 п.л.
Тираж 60 экз.

Центр оперативной печати «Оригинал 2»
г.Бердск, ул. Островского, 55, оф. 02, тел. (383) 214-45-35