ТЕХНОЛОГИЧЕСКОЕ ОБРАЗОВАНИЕ

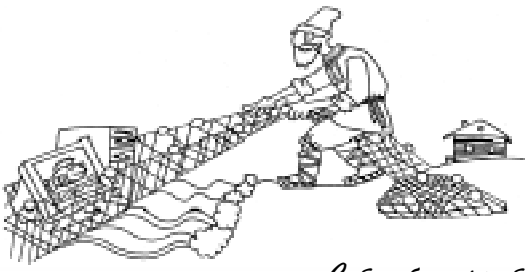
Соловьёв Игорь Павлович,
Усов Андрей Алексеевич

ВЗАИМОДЕЙСТВИЕ В СЕТИ ИНТЕРНЕТ И ЗАДАЧА ЕГО СПЕЦИФИКАЦИИ

ЗАДАЧА ПЕРЕДАЧИ ДАННЫХ, ЛОКАЛЬНАЯ СЕТЬ

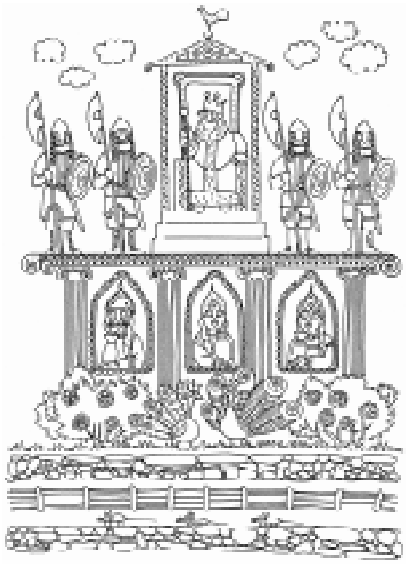
Современные информационные сети при ближайшем рассмотрении оказываются весьма сложными и неоднородными структурами, состоящими как из отдельных компьютеров, так и различных глобальных и локальных подсетей. Сети объединяют разнообразные вычислительные устройства (рабочие станции, серверы, принтеры и т. п.) – *узлы сети* (англ. – host), соединенные *линиями связи*, состоящими из кабелей, *сетевых адаптеров* и другого коммуникационного оборудования, работающего под управлением системного и прикладного программного обеспечения.

Не останавливаясь подробно на проблемах организации внутрисетевой передачи данных, обеспечиваемой специфическими для этой сети средствами физического и программного уровня, отметим лишь, что соответствующие программные механизмы, «надстроенные» над аппаратным слоем могут отличаться друг от друга не только по типу организации и физическим параметрам, но также и по производителю и различным платформам (Unix, OS/2, Win32, OS/390 (Mainframe) и другие). Очевидно, что без принятия всеми производителями оборудования и программного обеспечения общих правил и стандартов построения сетей прогресс в данной области был бы просто невозможен. Именно с этой целью были созданы международные комитеты и

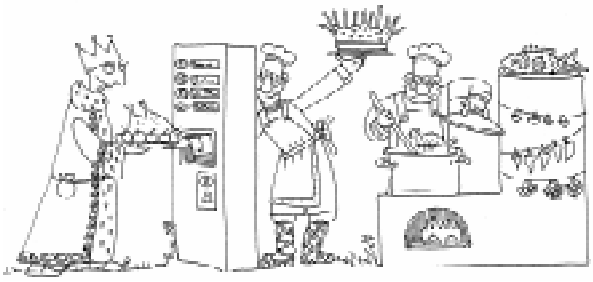


Сети объединяют разнообразные вычислительные устройства...

организации, в задачи которых входят анализ и стандартизация новых технологий, а также идеологических основ различных областей компьютерной отрасли. Так, в компьютерных сетях основой стандартизации стал «многоуровневый» подход. Именно этот подход, основывающийся на принципах



...в компьютерных сетях основой стандартизации стал «многоуровневый» подход...



...на соответствующих уровнях абстракции...

иерархической декомпозиции, послужил основой для выработки и принятия универсального подхода к организации взаимодействия сложных распределенных систем, широко используемый в индустрии в настоящее время.

ОТКРЫТЫЕ СИСТЕМЫ

В начале 80-х годов рядом международных организаций по стандартизации была разработана концепция организации распределенного взаимодействия так называемых *открытых систем* (Open System Interchanging, OSI), суть которой заключалась в построении иерархической структуры компонент сетевого взаимодействия. На каждом уровне такой структуры определяются основные функции, которые должны выполнять соответствующие ему компоненты, и *интерфейсы* – четко определенные правила взаимодействия с компонентами соседних уровней, использующие стандартизованные форматы сообщений.

В результате мы совершенно естественно приходим к иерархической структуре *коммуникационных слоев* сетевого взаимодействия, описывающих сложную распределенную систему на соответствующих *уровнях абстракции* (рисунок 1). Например, наиболее удобной для рассмотрения механизмов взаимодействия зачастую оказывается абстракция уровня приложений, например, «клиент-серверная» модель взаимодействия, не зависящая от конкретной реализации передачи данных на более низких уровнях. Поскольку иерархическая декомпозиция элементов системы позволяет отдельно рассматривать абстракцию организации взаимодействия любого уровня, наря-

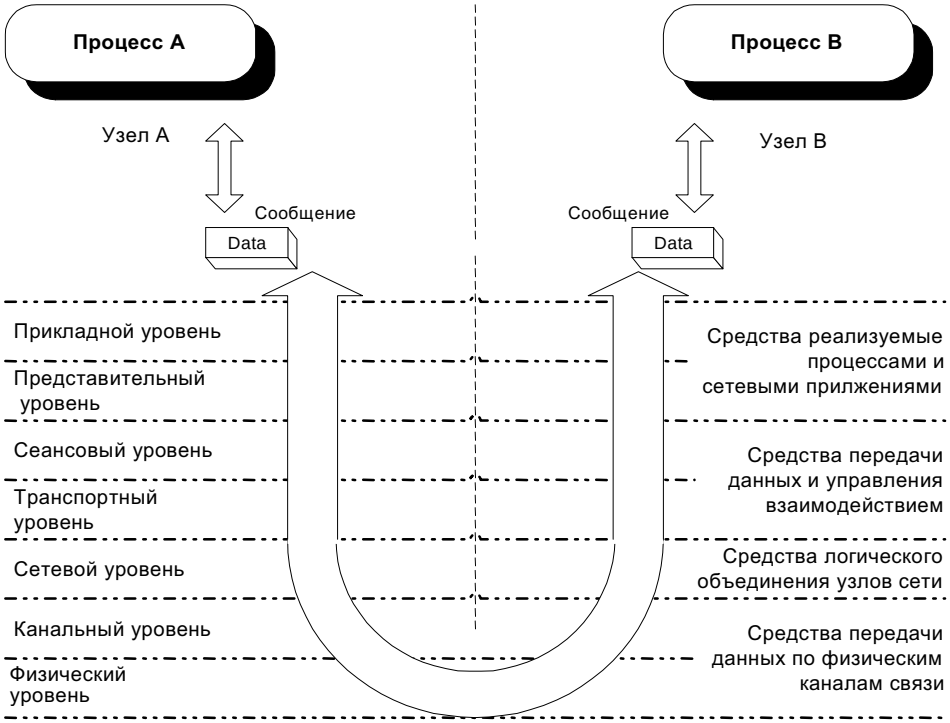


Рисунок 1. Обмен данными между процессами в концепции OSI.

ду с межуровневыми интерфейсами, модель OSI определяет *протоколы* – правила, устанавливающие порядок, формат и ограничения сообщений, которыми обмениваются компоненты одного уровня, находящиеся в разных узлах. Иерархически организованный набор протоколов, достаточный для организации взаимодействия в сети называют стеком *коммуникационных протоколов*. Мы рассмотрим типичный пример стека протоколов TCP/IP, организующих взаимодействие в сети Internet.

...отвлечься от особенностей низкоуровневой реализации и свободно использовать транспорт различных локальных сетей и физических подключений для передачи данных в протоколах более высокого уровня.

ГЛОБАЛЬНАЯ СЕТЬ ИЛИ «ИНТЕРНЕТ»

Лучшим доказательством успешности концепции открытых систем может служить ее быстрое распространение и широкое использование, в частности, как принцип построения сети Интернет.

Ввиду того, что сеть Интернет объединяет разнородные вычислительные сети, для успешного построения единой среды передачи данных необходимо решить проблемы, связанные с организацией взаимодействия между входящими в нее подсетями. Это достигается за счет унификации интерфейсов тех программных компонент, которые обеспечивают уровень связи с конкретной средой, специфичной для реализаций присоединенных к узлу сетей. Последнее позволяет производителям отвлечься от особенностей низкоуровневой реализации и свободно использовать транспорт различных локальных сетей и физических подключений для передачи данных в протоколах более высокого уровня.

Каждый узел может входить более чем в одну локальную сеть. Например, установив в компьютер два сетевых адаптера, подсоединенные, соответственно, к физическим линиям связи двух различных сетей, пользователь данного компьютера может использовать ресурсы обеих сетей одновременно. Однако этого еще не достаточно для настоящего объединения этих сетей. Для решения этой задачи некоторые узлы, снабженные необходимыми программными и аппаратными средствами, выделяются для обеспечения передачи данных между различными подсетями составной сети, в частности, – сети Интернет. В этом случае говорят, что они выполняют задачи *маршрутизации*. На рисунке 2 они представлены соединительными звеньями (R) между различными сетями, которые, в свою очередь, предоставляют специфичные для них средства внутрисетевой передачи данных, обеспечиваемые сетезависимым уровнем, обозначенным на рисунке штриховкой. Таким обра-

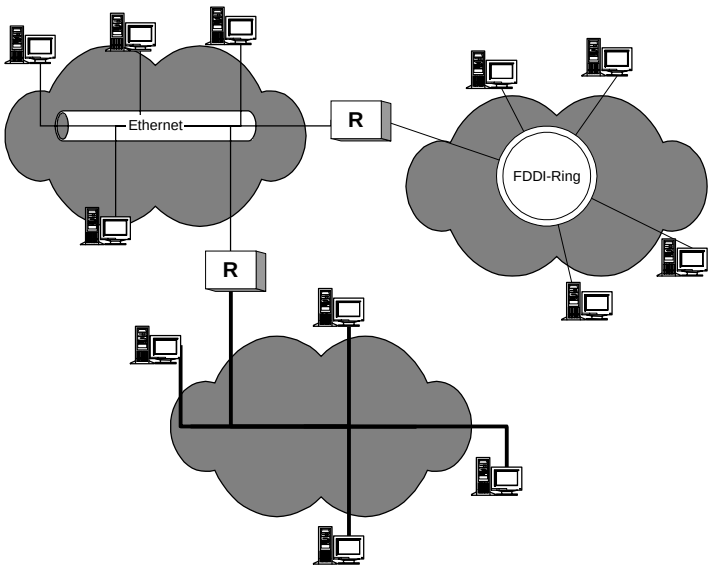
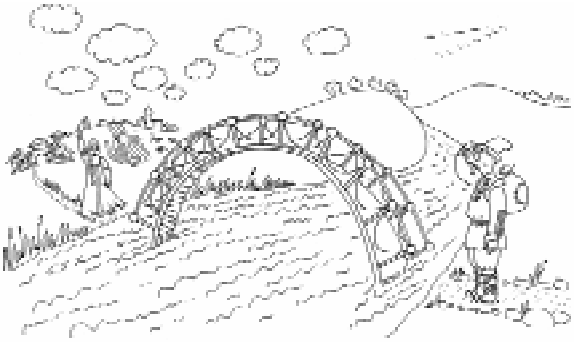


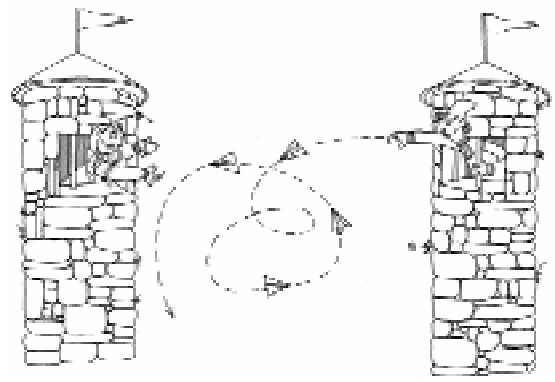
Рисунок 2. Составная сеть Интернет – объединения сетей различных типов.



...этого еще не достаточно для настоящего объединения этих сетей...

зом, можно говорить, что сетевой уровень координирует работу всех подсетей, лежащих на пути следования данных, используя в рамках подсетей характерные для них средства доставки данных.

Официальная документация, описывающая протоколы сети Интернет, изначально выделяет три важнейших уровня протоколов, совместно обеспечивающих функции доставки информации. Приложения, расположенные в сети для установления соединений и передачи данных, могут использовать *транспортный уровень* (Transport Layer) Интернет (соответствующий сеансовому и транспортному уровням OSI). Компоненты этого уровня обращаются, соответственно, к уровню *межсетевого взаимодействия* (Internetworking Layer) – сетевой уровень OSI, отвечающему за передачу данных между различными сетями. Последний, в свою очередь, использует компоненты уровня связи со средой (Link Layer), который занимает промежуточное положение между сетевым и канальным уровнями стандартной модели. Главной задачей уровня *связи* является предоставление стандартного интерфейса для доступа к средствам передачи данных, специфичным для различных используемых подсетей и физических подключений.



...на момент посылки сообщения дальнейший его путь труднопредсказуем...

ПРОБЛЕМА НАДЕЖНОСТИ ПРОЕКТИРОВАНИЯ

Глобальная сеть как совокупность программных и аппаратных средств является сложной распределенной по узлам сети системой, состоящей из огромного числа автономных подсистем, связанных между собой локальными сетями и низкоуровневыми средствами передачи данных. Для систем такой сложности важнейшими являются требования отказоустойчивости и адаптации к изменениям. Реализация этих требований в сети Интернет организована на уровне архитектуры всей системы. Так, для доставки *дейтаграммы* (пакета данных) по сети от одного пользовательского приложения другому необходимо также решить задачу, связанную с поиском маршрута пересылки. Таким образом, на момент посылки сообщения дальнейший его путь труднопредсказуем и алгоритмы поиска дальнейшего маршрута для дейтаграмм выполняются каждый раз заново при их пересылке от одного узла к другому. Кроме этого, протоколом межсетевого взаимодействия не гарантируется надежная доставка сообщений – эти функции возлагаются на протоколы более высокого уровня.

Требования надежности проектирования компонент сети Интернет, в частности, подразумевают наличие четко определенных алгоритмов, интерфейсов и протоколов взаимодействия. Однако стоит упомянуть, что с самого начала истории Интернет доступны лишь неформальные спецификации семейства протоколов этой сети, которые представлены в виде огромного числа текстовых документов (RFC, Request For Comments) [5] В частности, в ранних версиях этих документов встречаются неточности, послужившие причиной возникновения различных по смыслу реализаций программных компонент, реализующих описанные протоколы. Позднее появились документы, уточняющие алгоритмы и принци-

пы взаимодействия этих компонент, устраняя неоднозначности толкования некоторых из таких определений.

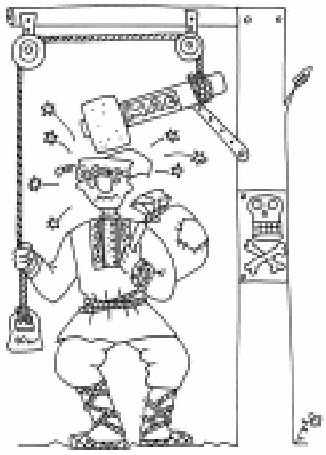
ФОРМАЛЬНЫЕ СПЕЦИФИКАЦИИ

Для определения того, как функционирует та или иная вычислительная система, или, другими словами, определения *операционной семантики*, проектировщики часто применяют различные неформальные или полуформальные методы описания или *спецификации*, такие как, например, графический язык UML или даже естественный язык. Подобные методы иногда приводят к двусмысленному толкованию различных аспектов функционирования проектируемой системы, в особенности, если речь идет о сложных распределенных или *мультиагентных* системах. В свою очередь, *формальные* методы позволяют применять точный математический аппарат как для описания поведения программных систем, так и в целях стандартизации и верификации (проверки алгоритмических свойств) систем.

Известно немало формальных методов спецификаций. Однако часто такие методы позволяют описывать системы на фиксированном, обычно низком уровне абстракции (неопытные программисты иногда даже думают, что спецификация – это программа, написанная ими на конкретном языке программирования).

Среди многих известных методов выделим один, базирующийся на понятии *Машины Абстрактных Состояний (МАС)*, или *Машины Гуревича*, – мощном средстве формальной спецификации операционной семантики программных систем на любом требуемом уровне абстракции (в англоязычной литературе, соответственно, используется термин *Abstract State Machine*).

Идея метода восходит к понятию *Машины Тьюринга* и сводится к описанию дискретной последовательности состояний S_0, S_1, ..., S_i, ... некоторой абстрактной вычислительной машины, моделирующей поведение заданного вычислительного процесса (процессов). Спецификация этого процесса записывается в виде *программы* на



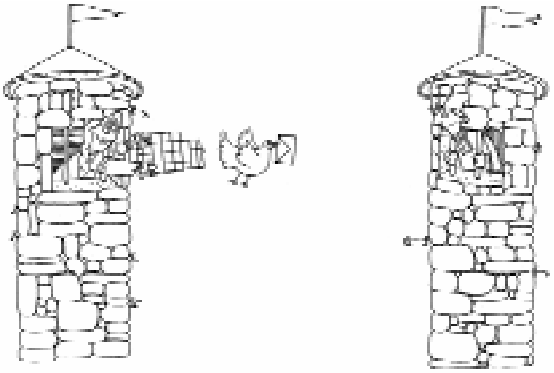
Машины Абстрактных Состояний (МАС)

довольно простом, но весьма общем языке высокого уровня, что позволяет легко освоить данный метод.

Каждая программа состоит из ограниченного набора *правил перехода* из одного состояния машины (S_i) в другое (S_{i+1}), причем базовые правила могут описывать некоторое изменение текущего состояния, ветвление программы или параллельные вычисления. Завершение работы описывается заданным логическим условием.

Важными особенностями метода являются ограниченность объема вычислений на каждом шаге, что, в отличие от многих известных формальных методов, позволяет моделировать поведение алгоритма, так сказать, «шаг в шаг», а также возможность последовательных уточнений и дополнений спецификации на любом требуемом уровне абстракций. Одной из самых интересных особенностей метода является возможность *исполнять* спецификации с помощью подходящего инструментального средства приблизительно так же, как выполняются и отлаживаются обычные программы. А наиболее желанной, вероятно, является возможность автоматической или полуавтоматической конвертации кода спецификации (полной или хотя бы частичной) в целевой код одного из обычных языков программирования, используемых для реализации вычислительных систем.

Для более детального изучения метода мы рекомендуем обратиться к [1], а также к посвященной данной тематике интер-



...одной из важных встроенных программных сущностей является агент — объект...

нет-странице на сайте мичиганского университета [3]. Здесь же мы проиллюстрируем применение данного метода на конкретном примере.

Заметим, что на практике обычно используется какая-либо конкретная реализация, расширяющая выразительные возможности языка метода MAC. Один из таких расширенных языков – объектно-ориентированный язык спецификаций AsmL – разработан в компании Microsoft [4]. Близкое по смыслу расширение языка, предназначенное для применения в инструментальной среде разработки и исполнения спецификаций мультиагентных систем, разработано авторами данной публикации [2].

ПРИМЕРЫ СПЕЦИФИКАЦИЙ

Одной из особенностей расширенного синтаксиса является поддержка многоуровневых («многослойных») спецификаций и «проектной» организации разработки спецификаций. Встроенные в язык специальные конструкции предназначены для описания семантики параллельных вычислений и анализа поведения сложных динамических систем, для которых подразумеваемой вычислительной моделью является модель мультиагентных систем. Так, одной из важных встроенных программных сущностей является агент – объект, инкапсулирующий поведение автономных компонент моделируемой системы (см. листинг 1)

Листинг 1.

```
class Agent
// определение базового понятия «агента»;

  send(msg as Object?, receiver as Agent)
  // процедура описывает механизм передачи данных (сообщений)между агентами
      if msg is not undef then
         MESSAGES(this, msg, receiver) := true

  receive(sender as Agent?) as Object?
  // выборка входящих сообщений - возвращает некоторый объект или undef
    if choose any from {true, false} then
    // недетерминированность выбора
       choose (snd, msg, rcv) from MESSAGES where \
         ((snd is sender) or (sender is undef)) \
          // выбор (по условию) элемента из множества MESSAGES
            and (rcv = this)
        MESSAGES(snd, msg, rcv) := false
         // удаление найденного элемента из множества
               return msg
               // возврат из процедуры
           ifnone return undef
            // если ничего не найдено - возвращаем undef
     Program()
     // главная программа агента (не детализируется в базовом классе);

// множество всех сообщений - тройка (Отправитель, Сообщение, Получатель):
var MESSAGES as Set of (Agent, Object, Agent)
```

Отметим, что в приводимых ниже примерах используется бесскобочная запись блока операций, записываемых вертикально, с одинаковым отступом, непосредственно друг под другом. Такие операции выполняются параллельно. Язык допускает запись одной операции в несколько строк, для чего в конце переносимой строки ставится «\».

По умолчанию предполагается недетерминированное частично-упорядоченное выполнение параллельных программ агентов с возможностью синхронизации (управления выполнением) программы, а также использован механизм недетерминированного взаимодействия (не синхронизированного по времени выполнения обмена сообщениями) между параллельно выполняемыми агентами.

УРОВНИ СПЕЦИФИКАЦИИ

В последующих примерах мы продемонстрируем как метод МАС позволяет естественным образом определять спецификации распределенных систем на различных уровнях абстракции, для чего рассмотрим фрагмент многоуровневой или «многослойной» спецификации Интернет-протоколов, а именно, системы организации межсетевого взаимодействия.

Каждый уровень содержит спецификацию определенных компонент распределенной системы сетевого взаимодействия и определяет семантику алгоритмов их работы. Все уровни спецификации именуются с помощью языковой конструкции **specification**. При этом, предполагается, что уровни спецификаций разрабатываются таким образом, что в результате они могут рассматриваться как самодостаточные и целостные определения выделенных слоев системы.

Ниже приводятся фрагменты нескольких уровней спецификации модели Интернет-протоколов. Уровень ***NetworkModel*** содержит описание базовых компонентов распределенной системы сетевого взаимодействия и расширяется межсетевой моделью взаимодействия ***InternetModel***. Спецификация ***ConnectionModel*** определяет модель передачи данных с установлением соединений. Фрагмент спецификации протокола транспортного уровня ***TCP_Protocol*** является примером определения межуровневого интерфейса, обеспечивающего взаимодействие между транспортным уровнем и уровнем приложений.

СЕТЕВАЯ МОДЕЛЬ ВЗАИМОДЕЙСТВИЯ

Рассмотрим сначала описание компонентов распределенной системы внутрисетевой передачи данных, в частности, – пример взаимодействия приложений, находящихся на разных узлах одной сети. В спецификации ***NetworkModel*** определим понятие сети (***Network***), объединяющей некоторое множество узлов (***hosts***), представленных абстрактным типом ***Host***, который здесь не детализируется, так же как и понятие адреса узла (***HostAddress***). Однако из приведенного фрагмента видно, что узлы могут входить более чем в одну сеть, в которой они уникально идентифицируются адресами, при этом уникальность адреса обеспечивается только в рамках одной сети (см. листинг 2).

Листинг 2.

```
specification NetworkModel // спецификация слоя базового определения сети

type Host                  // абстрактный тип - узел сети
type HostAddress           // абстрактный тип - адрес узла сети

class Network              // реализация понятия вычислительной сети
     var hosts as Set of Host     // переменная - множество узлов в сети
     var addressMap as Map of Address to Host   // таблица соответствий
                                                // адреса и узла

var NETWORKS as Set of Network = {}   // глобальная переменная -
                                      // множество всех сетей
```

Листинг 3.

```
specification NetworkModel   // продолжение спецификации…

class NetworkApplication extends Agent // - расширение базового класса
     var host as Host
    // переменная объекта – узел, на котором выполняется приложение
```

К этой же спецификации относится и определение сетевого приложения как абстрактного агента, выполняющегося на узле сети (см. листинг 3).

Взаимодействие приложений может осуществляться при условии использования общей среды передачи данных, то есть сети. Это ограничение может быть определено следующим образом (см. листинг 4).

Поскольку данный уровень абстракции не накладывает дополнительных ограничений на способ передачи данных по сети, тело процедуры посылки сообщения **send(...)** не детализируется в данной спецификации.

Далее расширим спецификацию ***NetworkModel*** за счет определения нового типа составной сети, представленного классом ***InternetNetwork***. Благодаря введению такого типа сети, приложения получают возможность передавать сообщения через составную сеть (в частности, к такому типу относится и сеть Интернет) (см. листинг 5).

ОРГАНИЗАЦИЯ ПЕРЕДАЧИ ДАННЫХ, ТРАНСПОРТНЫЙ УРОВЕНЬ

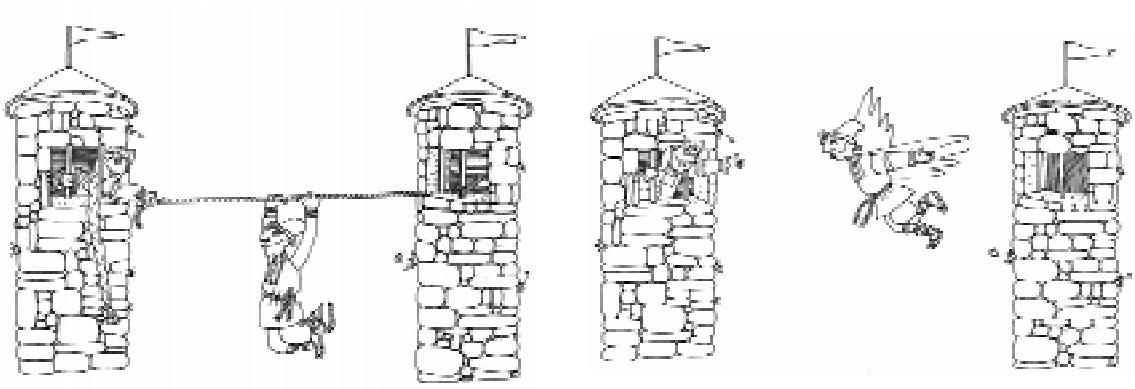
Принято различать два типа основных используемых протоколов – *с установлением соединения*, когда отправитель и получатель перед обменом данными должны сначала установить соединение, и *без предварительного установления соединения*, которые также иногда называют *дейтаграммными*. В большинстве случаев на

Листинг 4.

```
specification NetworkModel                  // продолжение спецификации…

class NetworkApplication extends Agent
// продолжение определения сетевого приложения
  ...
  send(msg as Object, receiver as Agent)
  // переопределение метода отправки сообщения
     require
            ((this in NetworkApplication) and \
            (receiver in NetworkApplication)) implies \
      isReachable((receiver as NetworkApplication).host, this.host)
    // - дополнительное ограничение на аргументы метода: для сетевых приложений
     // передача данных возможна только в рамках одной сети

/*Глобальная процедура, проверяющая, что два узла соединены какой-либо сетью.*/
isReachable(hostA as host, hostB as Host) as Boolean
  return hostA = hostB or \
```

Листинг 5.

```
specification InternetModel // спецификация модели сети Интернет
uses NetworkModel

class InternetNetwork extends Network
// составная сеть - частный случай сети
     var subnets as Set of Network
    // переменная класса – множество подсетей
```



...с установлением соединения... *...без предварительного установления соединения...*

уровне приложений используют протоколы первого типа.

Архитектура протоколов Интернет передает задачи установления соединения на транспортный уровень, к которому, в частности, относятся протоколы TCP (***Transmission Control Protocol***, протокол надежной доставки сообщений) и UDP (***User Datagram Protocol***, протокол дейтаграмм пользователя). Данные, поступающие на транспортный уровень, организуются системой в виде очередей сообщений к точкам входа приложений сетевых служб, которые называются портами и уникально идентифицируются в рамках одного узла (см. листинг 6 и рисунок 3).

Таким образом, сетевые службы прикладного уровня однозначно идентифицируются парами, состоящими из сетевого адреса и порта. Такую пару принято именовать *сокетом*.

Каждому соединению, устанавливаемому между двумя приложениями, соответствует пара сокетов, которые его уникально идентифицируют (см. листинг 7 и рисунок 4).

Наиболее распространенным примером организации взаимодействия является построение системы типа «клиент-сервер» на основе транспортного протокола TCP. Под сервером в данном случае понимают приложение (сетевую службу), принимающую входящие соединения от клиентских приложений.

Задачи, решаемые протоколом TCP, связаны с обеспечением гарантированной передачи данных между прикладными процессами (приложениями) на разных узлах.

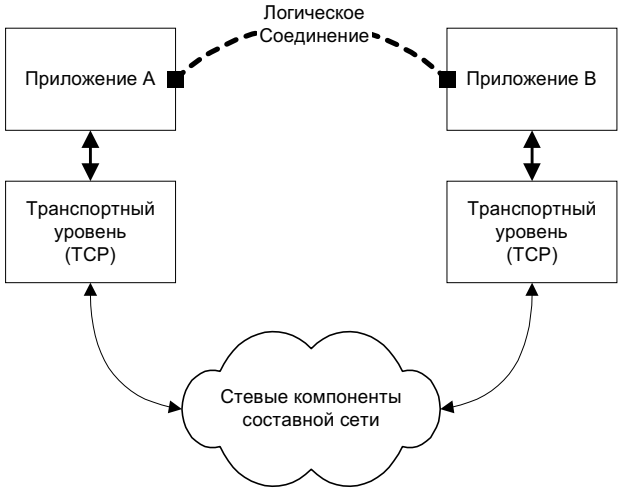


Рисунок 3. Организация потоков сообщений через логическое соединение.

Листинг 6.

```
specification ConnectionModel      // спецификация модели передачи данных
uses NetworkModel                  // детализация базовой модели

type Port                          // абстрактный тип - порт
class Socket
  var address as NetworkAddress    // переменная объекта – сетевой адрес
  var port as Port                 // переменная объекта – порт
```

Листинг 7.

```
specification ConnectionModel     // продолжение спецификации…
uses NetworkModel

type Stream                       // абстрактный поток данных
type ConnStatus                   // состояние соединения
type ConnName                     // идентификатор соединения

class Connection                  // класс реализующий понятие соединения
     var localSocket as Socket    // локальный сокет
     var foreignSocket as Socket? // удаленный сокет
     var inputQueue as Stream?    // входной поток (очередь) данных
     var outputQueue as Stream?   // выходной поток (очередь) данных
     var isOpen() as Boolean // функция: соед. открыто => true
     var name as ConnName     // системный идентификатор соединения
```

Построение исчерпывающей формальной спецификации протокола TCP является громоздкой задачей, в связи с этим мы здесь приводим фрагмент спецификации, описывающий межуровневый функциональный интерфейс протокола (см. листинг 8).

Конкретная реализация различных типов соединений и сокетов может значительным образом варьироваться. В частности, она зависит от особенностей реализации используемого соединением транспортного протокола.

Рассмотренное иерархическое построение формальной спецификации, основанное на использовании метода машин абстрактных состояний, предоставляет средства решения проблемы обеспечения надежности проектирования и реализации компонент системы. В то же время, применение такого инструментального средства, как интерпретатор спецификаций MAC, позволяет выполнять отладку архитектуры реализуемого модуля/модулей и производить динамическую верификацию в соответствии с определенными в спецификации условиями целостности на всех рассматриваемых уровнях абстракции системы. В частности, при рассмотрении межсетевой модели взаимодействия становится возможной отладка модели спецификаций протоколов Интернет на разнообразных *примерах реализации* механизмов передачи данных для различных типов подсетей и физических подключений.

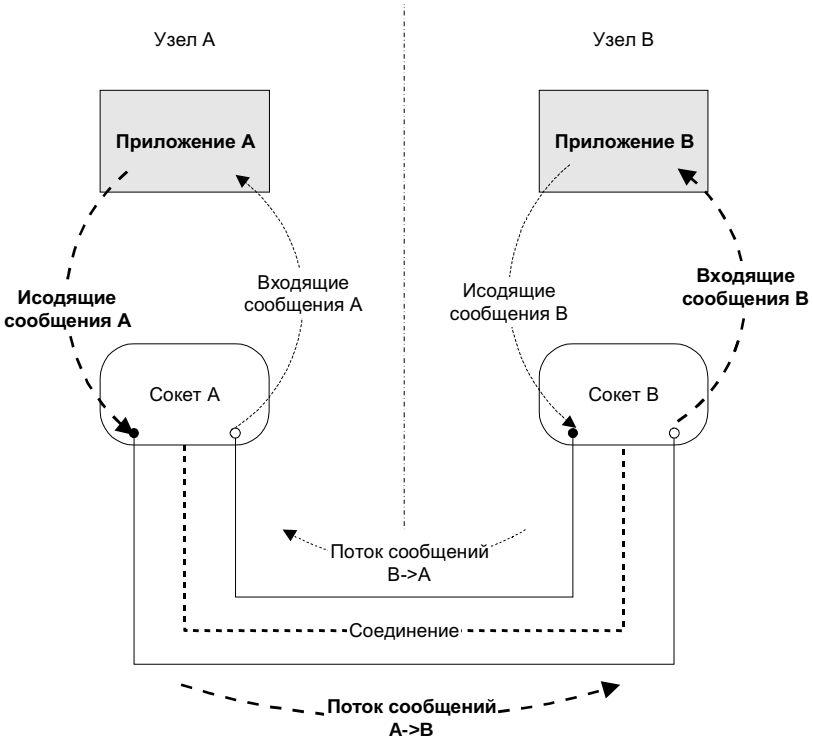


Рисунок 4. Обмен данными в случае использованием протокола с установлением соединения.

Листинг 8.

```
specification TCP_Protocol  // спецификация протокола TCP
uses ConnectionModel

class TCPModule extends NetworkApplication
// класс реализует программную компоненту транспортного уровня системы (TCP)

/* стандартные методы компоненты, реализующей протокол TCP*/
OPEN(localPort as Port, foreignSocket as Socket?, active as Boolean,
timeout as Integer?, options as TCPOptions?) as ConnName?

SEND(conn as ConnName, buffer as Buffer, PUSH as Boolean, URGENT as
Boolean, timeout as Integer)

RECEIVE(conn as ConnName, buffer as Buffer) as (Boolean, Boolean)

CLOSE(conn as ConnName)

STATUS(conn as ConnName) as ConnStatus

ABORT(conn as ConnName)
```

Литература.

1. *Соловьев И.П.* Формальные спецификации вычислительных систем. Машины Абстрактных Состояний (Машины Гуревича). Изд-во СПбГУ, 1998.

2. *Соловьев И.П., Усов А.А.* Проектирование и анализ мультиагентных систем с использованием интерпретатора машин абстрактных состояний. Материалы междисциплинарной научной конференции НБИТТ-21. Петрозаводск, 2002.

3. http://www.eecs.umich.edu/gasm

4. http://research.microsoft.com/fse

5. http://www.ietf.org/

Соловьев Игорь Павлович,
СПбГУ, мат.-мех. факультет,
доцент каф. информатики,

Усов Андрей Алексеевич,
СПбГУ, мат.-мех. факультет,
аспирант каф. информатики.