ИНСТИТУТ СИСТЕМНОГО ПРОГРАММИРОВАНИЯ РАН

ТЕХНИЧЕСКИЙ ОТЧЕТ

РАЗРАБОТКА ФОРМАЛЬНЫХ СПЕЦИФИКАЦИЙ И ТЕСТОВОГО НАБОРА ДЛЯ ПРОТОКОЛА С УСТАНОВЛЕНИЕМ СОЕДИНЕНИЯ ABRACADABRA

Москва 2008

Оглавление

1 Аннотация ... 3
2 Протокол ABRACADABRA ... 4
2.1 Взаимодействие протокола с пользователем ... 4
2.2 Взаимодействие между реализациями протокола ABRACADABRA ... 6
2.3 Установление соединения ... 7
2.4 Разрыв соединения ... 7
2.5 Передача данных ... 8
2.6 Процесс получения данных ... 8
2.7 Процесс отправки данных ... 8
2.8 Таблица переходов автомата протокола ABRACADABRA ... 9
3 Разработка тестового набора для протокола ABRACADABRA ... 12
3.1 Анализ исходной спецификации протокола ABRACADABRA ... 12
3.2 Формальная спецификация ABRACADABRA ... 12
3.2.1 Модельное состояние ... 13
3.2.2 Стимулы ... 14
3.2.3 Реакции ... 15
3.3 Медиаторы ... 16
3.4 Тестовые сценарии ... 17
3.5 Результаты тестирования ... 19
3.6 Особенности тестирования ABRACADABRA ... 19
4 Заключение ... 20
5 Список литературы ... 21
6 Приложения ... 22
6.1 Заголовочный файл формальной спецификации ... 22
6.2 Формальная спецификация протокола ABRACADABRA ... 24
6.3 Тестовый сценарий для тестирования соответствия спецификации протокола ABRACADABRA ... 34

1 Аннотация

В данной работе рассматриваются вопросы тестирования протоколов с установлением соединения с использованием технологии автоматизации тестирования UniTESK[1]. Для анализа применимости UniTESK к задаче тестирования соответствия реализаций протоколов с установлением соединения спецификациям таких протоколов разработан тестовый набор для «минимального» протокола с установлением соединения ABRACADABRA[2]. «Минимальность» протокола заключается в том, что данный протокол поддерживает все ключевые особенности рассматриваемого класса протоколов и, одновременно лишен сложных оптимизаций управления соединением и потоком данных, встречающихся в индустриальных протоколах, таких как TCP, RSCP и других. Протокол ABRACADABRA представляет собой упрощённый пример протокола с установлением соединения, которому, при всей его простоте, присущи все основные черты телекоммуникационных протоколов с установлением соединения. Протокол ABRACADABRA использовался для оценки применимости формальных методов (Formal Description Techniques, FDT) к формализации телекоммуникационных протоколов.

В отчете дано описание протокола ABRACADABRA, представлены формальные спецификации протокола на спецификационном расширении языка C и тестовые сценарии для проверки соответствия реализации протокола формальной спецификации.

2 Протокол ABRACADABRA

ABRACADABRA (Alternating Bit Protocol with Connection and Disconnection) – протокол с установлением соединения, который позволяет посылать данные в двух направлениях, используя Alternating Bit Protocol. Передача данных следует за соединением и прекращается после разъединения.

2.1 Взаимодействие протокола с пользователем

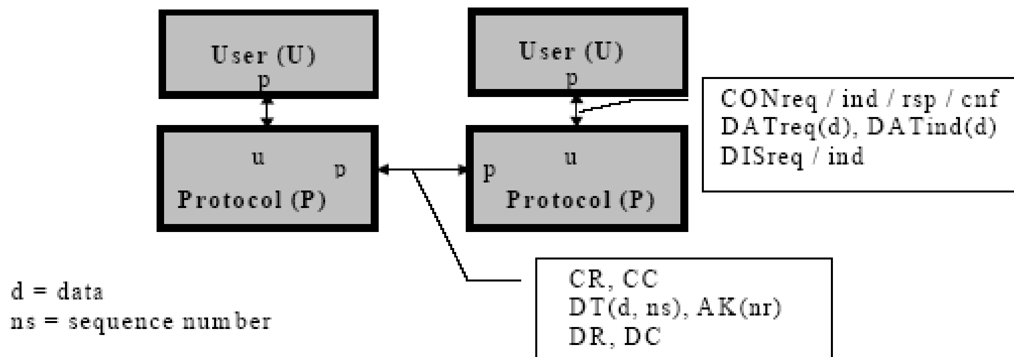
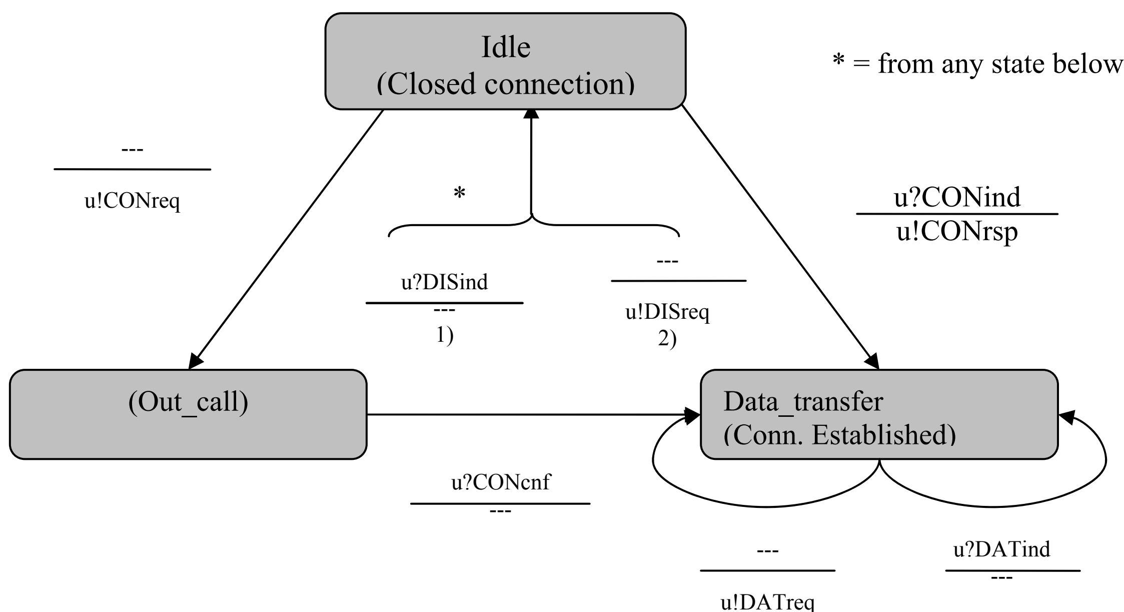


Рисунок 1 Применение протокола ABRACADABRA

Протокол ABRACADABRA предоставляет возможность пользователю установить соединение с другим пользователем и передавать данные по соединению в обе стороны. Соединение может быть разорвано любой из сторон в произвольный момент времени.

Протокол работает поверх ненадёжных сетей связи, так как для установления и поддержания соединения используются многочисленные повторы и подтверждения.

Рассмотрим автомат на рис.2., описывающий взаимодействие пользователя и реализации протокола. В серых прямоугольниках написаны названия состояний, на дугах записывают события, которые связаны с переходом. Запись на дуге состоит из двух частей, разделённых горизонтальной полосой. Над полосой пишется воздействие, которое оказывается на автомат извне, под полосой пишутся действия, которые производит автомат.



1)переход, инициализированный удалённым пользователем
2)переход, инициализированный локальным пользователем

Рисунок 2 Конечный автомат, описывающий взаимодействие пользователя и реализации протокола.

Изначально "пользователь" находится в состоянии (Idle) - соединение не установлено. Левая стрелка, обозначенная [-/u!CONreq], означает действие пользователя: пользователь (обозначен буквой u) выдаёт команду CONreq (это обозначается восклицательным знаком) реализации протокола. Символом CONreq обозначается команда "установить соединение". После этого пользователь переходит в состояние ”ожидание ответа” от реализации протокола. Это состояние обозначено как (Out_call).

Если в состоянии (Out_call) пользователь получит от реализации протокола подтверждение об установлении соединения, то пользователь переходит в состояние, в котором возможно передавать данные. Получение обозначается вопросительным знаком (стандартная нотация), подтверждение соединения в протоколе ABRACADABRA обозначается символом CONcnf. Состояние передачи данных разработчики диаграммы обозначили как (Data_Trasfer).

В состоянии (Data_Transfer) пользователь может получать данные [u?DATind] и отправлять данные [u!DATreq]. В состояние (Data_Transfer) пользователь может попасть из исходного состояния, если примет входящее сообщение об установлении соединения.

U?CONind

------------- означает, что пользователь получил сообщение о соединении [u?CONind] и

u!CONrsp

ответил приёмом сообщения о соединении [u!CONrsp].

В середине диаграммы обозначены два перехода, которые могут произойти в каждом состоянии, и оба ведут в начальное состояние. Первый переход происходит, когда пользователь получает сообщение о разрыве сообщения [u?DISind]. Второй переход - разрыв соединения самим пользователем - пользователь посылает сообщение DISreq [u!DISreq].

2.2 Взаимодействие между реализациями протокола ABRACADABRA

Рассмотрим автомат, моделирующий поведение реализации протокола.

Единицы обмена протокола (PDU):

Протокол ABRACADABRA насчитывает шесть единиц обмена протокола (PDU): CR, CC, DR, DC, DT, AK.

CR/CC - connect request/confirmation – запрос/подтверждение на соединение

DR/DC - disconnect request/confirmation – запрос/подтверждение на разъединение

DT (d, ns) = data packet – пакет с данными.

d = data vector (transmission - buffer: contents - SDU for sending) - данные (содержимое буфера передачи – данные пользователя для отправки)

ns = sender packet sequence number- номер посылаемого пакета

AK (nr) = positive acknowledgement – положительное подтверждение

nr = expected sequence number at reception – ожидаемый номер пакета на приеме

N - максимальное число попыток передачи пакета

На рис.3 дано упрощённое изображение автомата протокола ABRACADABRA. Помимо единиц обмена протокола на рисунке присутствуют несколько концептуальных переменных состояния («Концептуальные» в данном случае означает, что реализация не обязана использовать именно эти переменные, но внешне наблюдаемое поведение должно соответствовать поведению описанного ниже автомата):

Состояние протокола:

vs,vr – монотонно возрастающие переменные – номер пакета для отправителя, получателя соответственно (числа)

cnt - счетчик повторной передачи

xt = событие истечения таймера

При отправке пакетов в сеть часто используется параметр числа попыток передачи. В описании протокола ABRACADABRA этот параметр обозначается N.

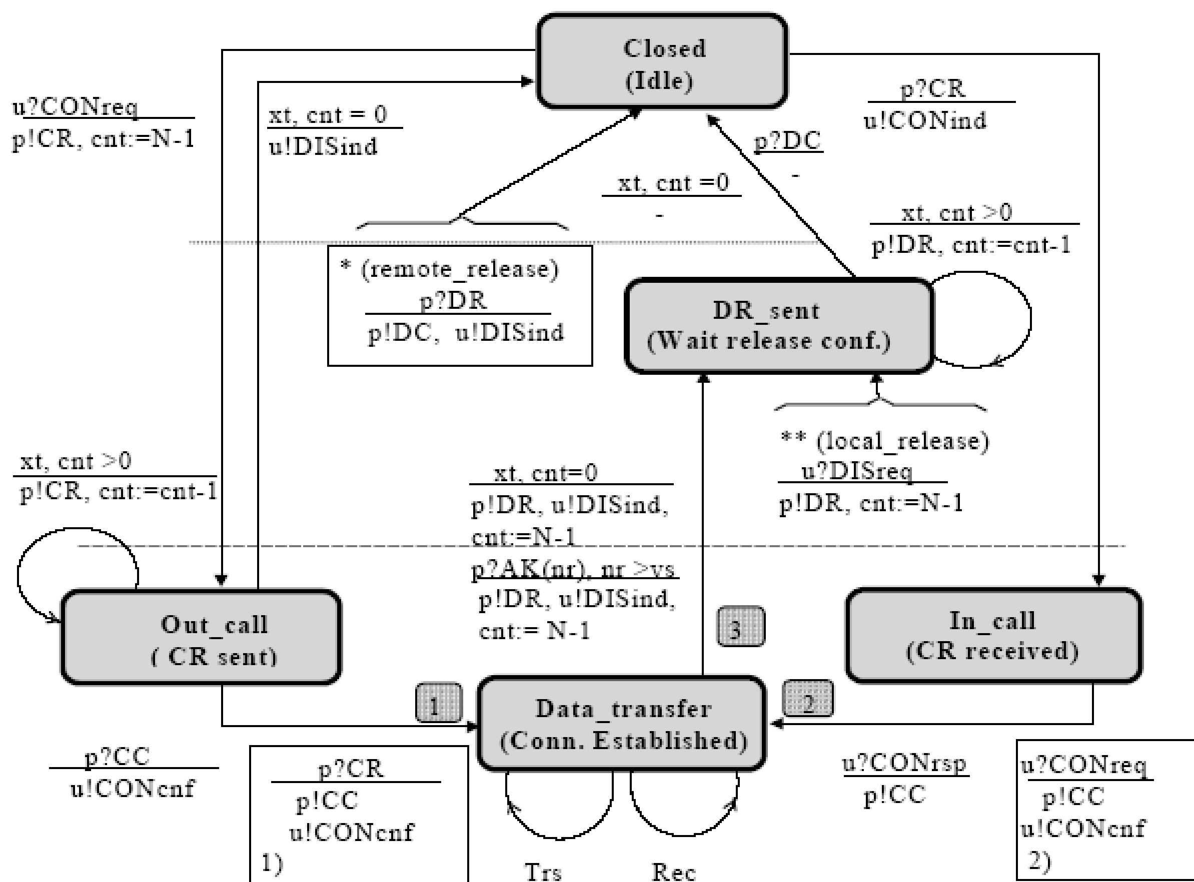


Рисунок 3 Автомат протокола ABRACADABRA (упрощённо)

2.3 Установление соединения

В протоколе ABRACADABRA участвуют две стороны. Одна сторона инициирует соединение, вторая сторона подтверждает соединение.

В начальный момент реализация находится в состоянии Closed(Idle), в котором нет соединения, все переменные имеют начальные значения. Когда пользователь запрашивает соединение сообщением CONreq, реализация протокола переходит в состояние ожидания Out_call. При переходе реализация отправляет запрос на соединение CR, устанавливает счётчик попыток установить соединение cnt равным N-1 и взводит таймер.

В состоянии Out_call реализация ожидает получить пакет CC – подтверждение об установлении соединения. Если подтверждение на соединение не приходит по истечении таймера, снова уменьшается cnt на единицу и если cnt > 0, то протокол еще раз посылает пакет CR. Если истек таймер и cnt = 0, то реализация протокола сообщает пользователю, что установить соединение не удалось, посылая сообщение DISind, и возвращается в начальное состояние Closed(Idle).

В состояние In_call реализация попадает, если приходит пакет с запросом на соединение CR (в этом случае пользователь получает запрос на соединение CONind). Протокол ждет ответ от пользователя.

Спецификация протокола разрешает пользователю только один ответ – согласие на соединение. Получив от пользователя согласие, протокол отсылает подтверждение (CC) и переходит в состояние Data_transfer.

2.4 Разрыв соединения

В состояние DR_sent протокол может попасть по двум причинам:

1) Пользователь пожелал разъединиться и послал сообщение DISind, протокол при этом посылает пакет DR. При этом взводится счетчик cnt и таймер. Если по истечении таймера не пришел пакет “подтверждение разъединения” DC, то счетчик уменьшается на 1 , взводится снова таймер и протокол снова посылает пакет DR. Если таймер истек и cnt=0, то снова протокол посылает пакет DR и пользователь переходит в начальное состояние.

2) Пользователь получил сообщение “запрос на разъединение” DISreq. Протокол посылает пакет DR, взводится таймер и cnt. Если по истечении таймера не пришел пакет “подтверждение разъединения” DC, то счетчик уменьшается на 1 , взводится снова таймер и протокол снова посылает DR. Если по истечении таймера cnt=0, то снова протокол посылает пакет DR и пользователь переходит в начальное состояние.

Если при cnt>0 протокол получил пакет DC, то пользователь переходит в начальное состояние Closed(Idle).

2.5 Передача данных

В состояние Data_transfer (Conn. Established) пользователь попадает, если пришел пакет CC в состоянии Out_call (пользователь получил сообщение CONcnf - Connection confirmation) или пользователь был в состоянии In_call и дал разрешение на соединение CONrsp (протокол посылает пакет CC). Устанавливается соединение. Когда установлено соединение между двумя протоколами, в это время больше никто не может подключиться. Передача данных может быть как однонаправленной, так и двунаправленной, в зависимости от желания пользователя.

Рассмотрим подробней состояние отправки и передачи данных Data_transfer (рис.4).

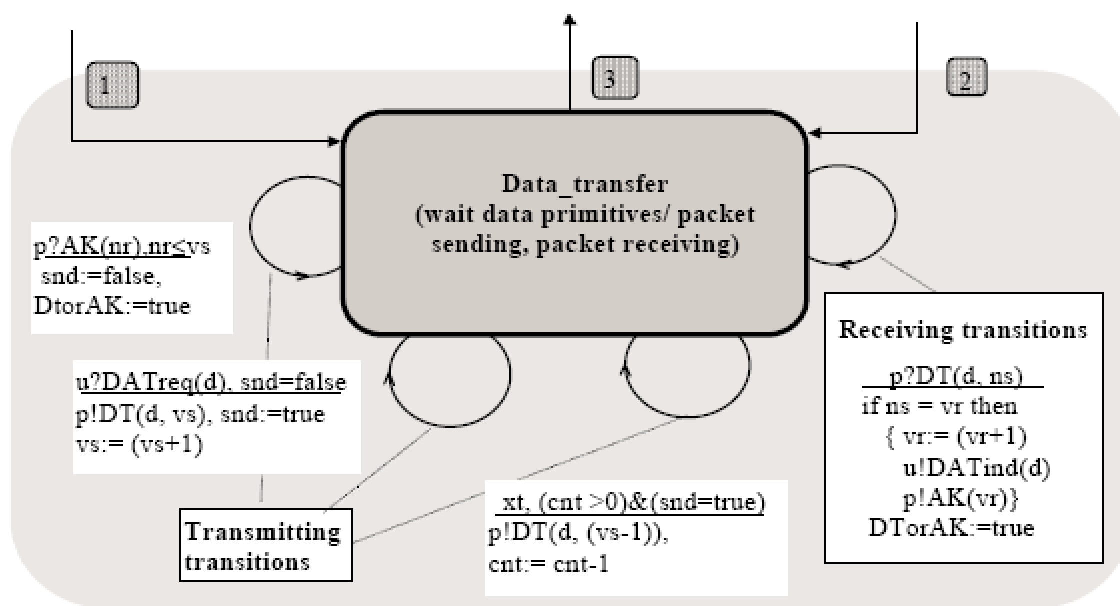


Рисунок 4 Приём / передача данных

2.6 Процесс получения данных

Протокол получает пакет с данными DT(d,ns), где d – данные, ns – номер пакета. Если ns= vr (где vr – номер последнего пакета (который был принят протоколом- получателем) + 1), то увеличиваем vr на 1, посылаем пользователю полученные данные DATind(d), а протоколу- отправителю посылаем пакет с подтверждением AK(vr).

2.7 Процесс отправки данных

Если протокол получает пакет с подтверждением получения данных другим протоколом AK(nr), и nr<vs, где vs - номер последнего пакета (который был послан протоколом -

отправителем) + 1), то устанавливаем snd := false, где snd – флаг, предупреждающий, что на данный момент данные не посылаются и что все предыдущие данные успешно доставлены.

Если пользователь сообщил протоколу, что он хочет послать данные DATreq(d) и при этом флаг snd = false, то протокол посылает пакет с данными DT(d,vs), устанавливает флаг snd := true, и увеличивает счетчик vs на 1. Также взводится таймер xt и счетчик cnt. Если истекает таймер и (cnt>0)&(snd=true), то заново посылается пакет DT(d, (vs-1)) и счетчик cnt уменьшается на 1.

2.8 Таблица переходов автомата протокола ABRACADABRA

В следующей таблице представлены переходы автомата в виде таблицы.

Каждый переход автомата представлен в отдельной строке таблицы. В описании перехода указываются

- начальное состояние перехода,
- стимул, по которому совершается переход,
- условие совершения перехода – логическое условие, выраженное формулой или текстом на естественном языке,
- действие перехода – изменение состояния, выраженное формальными присваиваниями или текстом на естественном языке,
- реакция, выдаваемая протоколом – значения параметров реакций задаются в виде формул или текстом на естественном языке,
- конечное состояние,
- примечания.

Условия перехода и действия описываются как условия или операции с состоянием автомата. В таблице переходов ABRACADABRA состояние протокола задаётся следующими переменными:

- vs,vr – монотонно возрастающие переменные – номер пакета для отправителя, получателя соответственно (числа)
- cnt - счетчик повторной передачи

В таблице используются следующие обозначения для событий и сообщений:

- xt – событие истечения таймера
- u!XX – Сообщение XX отсылается протоколом на верхний уровень(уровень пользователя)
- u?XX – Верхний уровень (пользователь) передаёт протоколу сообщение XX.
- p!YY – Протокол получает с нижнего уровня (от другой реализации протокола) сообщение YY
- p?YY – Протокол отправляет на нижний уровень сообщение YY.

В таблице используется следующий алфавит для представления стимулов и реакций протокола ABRACADABRA в обменах данными с верхним уровнем:

- CONreq – запрос пользователя на установление соединение.
- CONind - уведомление протоколом пользователя об установленнии соединении.
- CONrsp - разрешение пользователем установить входящее соединение.
- CONcnf - подтверждение установления соединения от протокола пользователю.
- DISreq - запрос пользователя на разрыв соединения.
- DISind – уведомление протоколом пользователя о разрыве соединения.

- DATreq(d) – запрос пользователя на передачу данных. Данные задаются значением параметра d.
- DATind(d) – доставка протоколом пользователю входящих данных. Полученные данные задаются значением параметра d.

Обмен сообщениями между реализациями протокола ABRACADABRA задаётся следующим алфавитом:

- **CR/CC** - connect request/confirmation – запрос/подтверждение соединения
- **DR/DC** - disconnect request/confirmation – запрос/подтверждение разъединения
- **DT (d, ns) -** data packet – пакет с данными. Параметры сообщения:
    - d - data vector (transmission - buffer: contents - SDU for sending) - данные (содержимое буфера передачи – данные пользователя для отправки)
    - ns - sender packet sequence number- номер посылаемого пакета
- **AK (nr)** = positive acknowledgement – положительное подтверждение получения данных. Параметры сообщения:
    - nr - expected sequence number at reception – ожидаемый номер пакета на приеме

Таблица 1 Таблица переходов автомата протокола ABRACADABRA

Начальное состояние	символ алфавита	предусловие	действие	реакция	кон. Состояние	Примечание
Closed	p?CR			u!CONind	In_call	
Closed	u?CONreq		cnt:= N-1; взводится таймер	p!CR	Out_call	
Data_transfer	p?AK(nr)	nr>vs	cnt:= N-1; взводится таймер	p!DR,u!DISind	DR_sent	
Data_transfer	p?AK(nr)	nr=vs	snd:=false; сбросить таймер		Data_transfer	
Data_transfer	p?AK(nr)	nr<vs	snd:=false; сбросить таймер		Data_transfer	
Data_transfer	p?CR		???	???		Проблема протокола: нет перехода по получению CR в состоянии Data_transfer.
Data_transfer	p?DR			p!DC,u!DISind	Closed	
Data_transfer	p?DT(d,ns)	ns=vr	vr:=vr+1	u!DATind(d), p!AK(post(vr))	Data_transfer	
Data_transfer	p?DT(d,ns)	ns=vr-1		p!AK(vr)	Data_transfer	
Data_transfer	p?DT(d,ns)	ns!=vr && ns!=vr-1			Data_transfer	
Data_transfer	u?DATreq(d)	snd=false	vs:=vs+1,snd:=true; cnt:= N-1; взводится таймер	p!DT(d,pre(vs))	Data_transfer	
Data_transfer	u?DISreq		cnt:=N-1	p!DR	DR_sent	
Data_transfer	Таймаут	cnt=0	Взводится таймер,cnt:= N-1	p!DR,u!DISind	DR_sent	
Data_transfer	Таймаут	(cnt>0) AND (snd=true)	Взводится таймер,cnt:=cnt-1	p!DT(d,vs-1)	Data_transfer	
DR_sent	p?DC		сбрасывается таймер		Closed	
DR_sent	p?DR			p!DC,u!DISind	Closed	В этом переходе нет потребности передавать u!DISind, так как эта реакция уже была выдана при переходе в DR_Sent. Протокол ведь не выдаёт DISInd при переходе по p?DC. Этот переход можно рассматривать как неточность спецификации.
DR_sent	Таймаут	cnt>0	Взводится таймер,cnt:=cnt-1	p!DR	DR_sent	
DR_sent	Таймаут	cnt=0			Closed	
In_call	p?DR			p!DC,u!DISind	Closed	
In_call	u?CONreq			p!CC,u!CONcnf	Data_transfer	
In_call	u?CONrsp			p!CC	Data_transfer	
In_call	u?DISreq		cnt:= N-1; взводится таймер	p!DR	DR_sent	
In_call	p?CR	???	???	???	???	
Out_call	p?CC		сбрасывается таймер	u!CONcnf	Data_transfer	
Out_call	p?CR		сбрасывается таймер	p!CC,u!CONcnf	Data_transfer	
Out_call	p?DR		сбрасывается таймер	p!DC,u!DISind	Closed	
Out_call	u?DISreq		cnt:= N-1; взводится таймер	p!DR	DR_sent	
Out_call	Таймаут	cnt>0	Взводится таймер,cnt:=cnt-1	p!CR	Out_call	
Out_call	Таймаут	cnt=0		u!DISind	Closed	

3 Разработка тестового набора для протокола ABRACADABRA

В данной главе рассмотрена формальная спецификация протокола ABRACADABRA и представлен результат работ по созданию тестового набора для тестирования соответствия спецификации протокола ABRACADABRA.

3.1 Анализ исходной спецификации протокола ABRACADABRA

В работе требования были сведены в таблицу, которая явно задаёт переходы автомата протокола ABRACADABRA.

CR/CC - connect request/confirmation – запрос/подтверждение на соединение
DR/DC - disconnect request/confirmation – запрос/подтверждение на разъединение
DT (d, ns) = data packet – пакет с данными.
d = data vector (transmission - buffer: contents - SDU for sending) - данные (содержимое буфера передачи – данные пользователя для отправки)
ns = sender packet sequence number- номер посылаемого пакета
AK (nr) = positive acknowledgement – положительное подтверждение
nr = expected sequence number at reception – ожидаемый номер пакета на приеме
N - максимальное число попыток передачи пакета
vs,vr – монотонно возрастающие переменные – номер пакета для отправителя, получателя соответственно (числа)
cnt - счетчик повторной передачи
xt = событие истечения таймера
u! – Сообщение отсылается протоколом на верхний уровень(уровень пользователя)

u? – Сообщение протоколу с верхнего уровня (от пользователя)

p! – Сообщение, полученное протоколом с нижнего уровня (с link-a)

p? – Сообщение , отправненное протоколом в link CONreq – запрос пользователя на установление соединение

DATreq(d) – пользователь просит передать данные

DISreq - запрос пользователя на разрыв соединения

CONrsp - разрешение пользователем установления соединения

CONind - уведомление об установленном соединении пользователю

DISind – уведомление о разрыве соединения пользователю

DATind(d) – передача пользователю присланных данных

CONcnf - подтверждение установления соединения от протокола пользователю

CONreq – запрос пользователя на установление соединение

3.2 Формальная спецификация ABRACADABRA

Спецификация включает 10 стимулов и 10 реакций. Стимулы и реакции моделируют обработку входящих и исходящих сообщений протокола и изменения состояния протокола.

3.2.1 Модельное состояние

Модельное состояние протокола состоит из идентификатора текущего состояния (CLOSED, OUT_CALL и т.д.), текущих значений параметров и флагов ожидаемых реакций.

Рассмотрим переменные модельного состояния:

Таблица 2 Переменные модельного состояния

Переменная/функция	Тип	Назначение
current_state	state_name_t	Символ текущего состояния протокола: CLOSED, OUT_CALL, IN_CALL, DATA_TRANSFER, DR_SENT
current_state	state_name_t	текущее состояние
vs	int	номер пакета отправителя
vr	int	номер пакета получателя
max_retries_param	int	максимальное число попыток
cnt	int	счетчик повторной передачи
should_send_CR	bool	флаг = true, если следует выслать CR
should_send_CC	bool	флаг = true, если следует выслать CC
should_send_DR	bool	флаг = true, если следует выслать DR
should_send_DC	bool	флаг = true, если следует выслать DC
should_send_CONcnf	bool	флаг = true, если следует выслать CONcnf пользователю
should_send_CONind	bool	флаг = true, если следует выслать CONind пользователю
should_send_DISind	bool	флаг = true, если следует выслать DISind
should_send_DATind	bool	флаг = true, если следует выслать DATind
should_send_ACK	bool	флаг = true, если следует выслать ACK
should_send_DT	bool	флаг = true, если следует выслать DT
sending	bool	флаг = true, если отправляются данные, и пока не было подтверждения

Вследствие ограничений инструмента CTesK переменные модельного состояния, в которых хранятся отправленные и принятые данные, не доступны для прямого присваивания. По этой причине они задаются как пары функций:

- Данные, отправленные пользователем через стимул DATreq, моделируются как список байтов, тип List. Для работы с этой переменной в модель включаются две функции:
    - List * get_data_sent() – получить текущее значение переменной.
    - void set_data_sent(List * value) – установить значение переменной.
- Данные, полученные протоколом из сети стимул DT, моделируются как список байтов, тип List. Для работы с этой переменной в модель включаются две функции:
    - List * get_data_received() – получить текущее значение переменной.

- void set_data_received(List * value) – установить значение переменной.

Кроме того, для работы с символом состояния в спецификацию включены две вспомогательные функции, которые возвращают строковое имя состояния:

const char * get_state_name(int state_id) – возвращает имя заданного состояния state_id.

const char * get_current_state_name() – возвращает имя текущего состояния.

Пара функций get_state_name и get_current_state_name возвращают строку с именем состояния (“CLOSED”, “OUT_CALL”, и т.д.). Память под возвращаемую строку отводится статически, поэтому освобождать память после вызова функций нельзя.

3.2.2 Стимулы

В этом разделе перечислены стимулы протокола. Протокол принимает 4 видов стимулов от пользователя и 6 видов сообщений протокола из сети. Каждому виду стимулов соответствует отдельная спецификационная функция.

Имена стимулов в спецификации основываются на именах сообщений в исходной спецификации протокола.

Для сообщения пользователя, например CONreq, имя соответствующей спецификационной функции получается добавлением суффикса _spec: CONreq_spec.

void CONreq_spec(void)

Таблица 3 Стимулы

Стимулы	Описание
CONreq_spec(void)	Стимул CONreq_spec моделирует сообщение пользователя CONreq. Пользователь передаёт сообщение CONreq для того, чтобы протокол установил соединение с другим узлом.
CONrsp_spec(void);	Стимул CONrsp_spec моделирует сообщение пользователя CONrsp. Пользователь передаёт сообщение CONrsp, тем самым разрешая соединение с другим узлом.
DISreq_spec(void);	Стимул DISreq_spec моделирует сообщение пользователя DISreq. Пользователь передаёт сообщение DISreq для того, чтобы протокол разорвал соединение с другим узлом.
DATreq_spec(List * bytes);	Стимул DATreq_spec моделирует сообщение пользователя DATreq. Пользователь передаёт сообщение DATreq для того, чтобы протокол передал данные bytes на другую сторону.
receive_CR_spec(void);	Стимул receive_CR_spec моделирует сообщение протокола CR, отправленное другой стороной. Тем самым другая сторона сообщает о том, что она хочет подключиться.
receive_CC_spec(void);	Стимул receive_CC_spec моделирует сообщение протокола CC, отправленное другой стороной. Тем самым другая сторона сообщает о том, что соединение установлено.
receive_DR_spec(void);	Стимул receive_DR_spec моделирует сообщение протокола DR, отправленное другой стороной. Другая

	сторона сообщает о том, что она хочет отключиться.
receive_DC_spec(void);	Стимул receive_DC_spec моделирует сообщение протокола DC, отправленное другой стороной. Тем самым другая сторона сообщает о том, что соединение разорвано.
receive_DT_spec(DT_PDU * dt);	Стимул receive_DT_spec моделирует сообщение протокола DT, отправленное другой стороной. Другая сторона послала данные dt.
receive_ACK_spec (ACK_PD * ack);	Стимул receive_ACK_spec моделирует сообщение протокола ACK, отправленное другой стороной. Другая сторона подтверждает получение данных, отправленных протоколом.

3.2.3 Реакции

В этом разделе перечислены реакции протокола. Протокол передает 4 вида сообщений пользователю и отсылает 6 видов сообщений протокола в сеть. Каждому виду реакций соответствует отдельная спецификационная функция.

Имена реакций в спецификации основываются на именах сообщений в исходной спецификации протокола.

В спецификационном расширении языка Си требуется, чтобы у реакции всегда было возвращаемое значение. Большинство реакций в исходной спецификации протокола не имеют параметров. Такие реакции моделируются в спецификации как возвращающие объект типа Unit.

Для сообщения пользователя, например CONind, имя соответствующей спецификационной функции получается добавлением суффикса _spec: CONind_spec.

Таблица 4 Реакции

Реакции	Описание
CONind_spec(void)	Реакция CONind_spec моделирует сообщение пользователю CONind. Протокол передаёт сообщение CONind пользователю, сообщая, что соединение с другим узлом установлено.
CONcnf_spec(void)	Реакция CONcnf_spec моделирует сообщение пользователю CONcnf. Протокол передаёт сообщение CONcnf пользователю, подтверждая то, что соединение с другим узлом установлено.
DISind_spec(void)	Реакция DISind_spec моделирует сообщение пользователю DISind. Протокол передаёт сообщение DISind пользователю, сообщая, что соединение с другим узлом разорвано.
DATind_spec(void)	Реакция DATind_spec моделирует сообщение пользователю DATind. Протокол передаёт сообщение DATind пользователю, которое представляет собой данные, отправленные другой стороной.
send_CR_spec(void)	Реакция send_CR_spec моделирует сообщение протокола CR. Протокол передаёт сообщение CR

	другой стороне, сообщая о том, что необходимо установить соединение.
send_CC_spec(void)	Реакция send_CC_spec моделирует сообщение протокола CC. Протокол передаёт сообщение CC другой стороне, сообщая о том, что соединение установлено.
send_DR_spec(void)	Реакция send_DR_spec моделирует сообщение протокола DR. Протокол передаёт сообщение DR другой стороне, сообщая о том, что необходимо разорвать соединение.
send_DC_spec(void)	Реакция send_DC_spec моделирует сообщение протокола DC. Протокол передаёт сообщение DC другой стороне, подтверждая разрыв соединения.
send_DT_spec(void)	Реакция send_DT_spec моделирует сообщение протокола DT, в котором передаются данные другой стороне.
send_ACK_spec(void)	Стимул send_ACK_spec моделирует сообщение протокола ACK, отправленное другой стороне. Протокол подтверждает получение данных, отправленных другой стороной.

3.3 Медиаторы

Для тестирования используется специальная версия протокола нижнего уровня, которая

- перехватывает отсылаемые сообщения и регистрирует их как реакции
- отправляет тестовые воздействия в реализацию

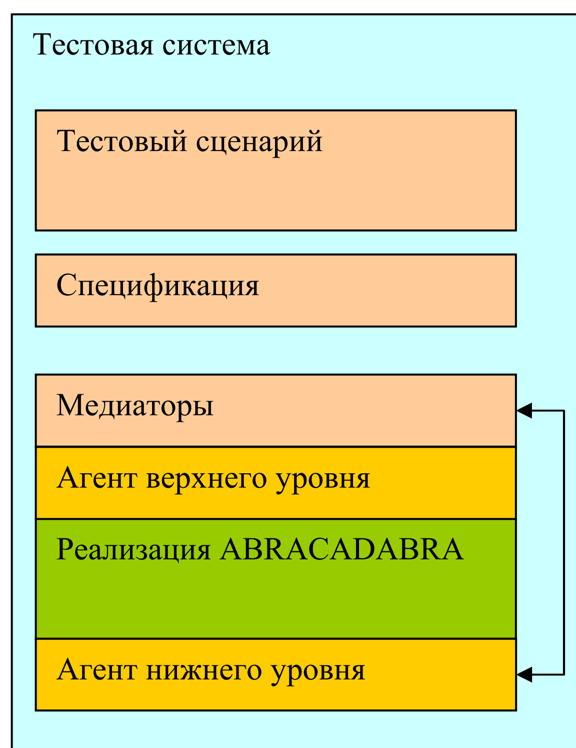


Рисунок 5 Устройство тестового стенда

Медиаторы:

CONreq_media

CONrsp_media;

CONind_media;

CONcnf_media;

DISreq_media;

DISind_media;

DATreq_media;

DATind_media;

receive_CR_media;

send_CR_media;

receive_CC_media;

send_CC_media;

receive_DR_media;

send_DR_media;

receive_DC_media;

send_DC_media;

receive_DT_media;

send_DT_media;

receive_ACK_media;

send_ACK_media.

3.4 Тестовые сценарии

В рамках данной работы был разработан 1 тестовый сценарий, который содержит 12 сценарных функций. В совокупности сценарные функции обеспечивают 97% покрытие спецификации.

Всего в спецификации 10 стимулов насчитывается 25 элементов покрытия, из которых тестовый сценарий покрывает 24. Все 10 реакций, описанных в спецификации, встречаются при работе сценария.

Время работы сценария 3 мин. Автомат сценария насчитывает три состояния и 31 переход.

В таблице 2 представлены краткие описания сценарных функций.

Таблица 5 Сценарные функции для тестирования протокола ABRACADABRA

Сценарная функция	Описание
outcall_scen;	Сценарная функция тестирует установление соединения по запросу пользователя. Сценарная функция выдаёт стимул CONreq в состоянии CLOSED и итерирует различные варианты подтверждений установления соединения: 1. Отсутствие подтверждения на запрос о соединении. 2. Итерация интервала времени между выдачей реакции CC (подтверждение соединения) и подачей в реализацию подтверждения об

	установлении соединения
outcall_with_incall_scen;	Сценарная функция тестирует установление соединения по запросу пользователя. Сценарная функция выдаёт стимул CONreq в состоянии CLOSED и итерирует различные варианты запроса на установление соединения: 3. Отсутствие запроса о соединении. 4. Итерация интервала времени между выдачей реакции CR (запрос на соединение) и подачей в реализацию подтверждения об установлении соединения
data_transfer_send_scen;	Сценарная функция тестирует передачу данных по запросу пользователя. Сценарная функция выдаёт стимул DATreq и итерирует различные варианты подтверждений о получении данных другой стороной
data_transfer_recieve_scen;	Сценарная функция тестирует получение данных, отправленных другой стороной. Сценарная функция выдаёт стимул DT и итерирует различные варианты получения данных: 1) Пакет с правильным номером 2) Пакет, полученный второй раз 3) Пакет, с неожидаемым номером
incall_scen;	Сценарная функция тестирует запрос на соединение по запросу второй стороны. Сценарная функция выдаёт стимул CR в состоянии CLOSED.
incall_accept;	Сценарная функция тестирует подтверждение соединения от пользователя. Сценарная функция выдаёт стимул CONrsp.
incall_conreq;	Сценарная функция тестирует запрос на соединение по запросу пользователя. Сценарная функция выдаёт стимул CONreq в состоянии CLOSED.
user_disconnect_scen;	Сценарная функция тестирует разрыв соединения по запросу пользователя. Сценарная функция выдаёт стимул DISreq и итерирует различные варианты подтверждений разрыва соединения
cancel_outcall_scen.	Сценарная функция тестирует разрыв соединения по запросу пользователя в состоянии OUTCALL. Сценарная функция выдаёт стимул CONreq в состоянии CLOSED, затем выдает запрос о разрыве соединения от пользователя.

outcall_dr_scen	Сценарная функция тестирует разрыв соединения по запросу другой стороны. Сценарная функция выдаёт стимул CONreq в состоянии CLOSED и итерирует различные варианты запроса разрыва соединения.
disconnect_request_scen	Сценарная функция тестирует запрос о разрыве соединения другой стороны. Сценарная функция выдаёт стимул DR в состоянии DATA_TRANSFER или INCALL.
user_disconnect_with_dr_scen	Сценарная функция тестирует разрыв соединения по запросу пользователя в тот момент, когда другая сторона присылает сообщение DR. Сценарная функция выдаёт стимул DISreq и итерирует различные варианты запроса о разрыве соединения

3.5 Результаты тестирования

Выявлены и исправлены ошибки в реализации.

Исправлены ошибки в спецификации:

1. В состоянии data_transfer реализация не может получать сообщение CR. Это может привести к тому, что вторая сторона не сможет установить соединение. Например: ответ CC не дошёл до инициатора соединения. Тогда спустя таймаут инциатор снова отправит CR, но на этот раз ответчик, находясь уже в состоянии data_transfer, не сможет на него ответить, так как в состоянии Data_transfer не запланировано отвечать на CR.
2. При анализе протокола обнаружено, что при получении предыдущего пакета реализация не реагирует. Необходимо в этом случае посылать подтверждение.
3. В состоянии dr_sent по стимулу DR нет потребности передавать u!DISind, так как эта реакция уже была выдана при переходе в DR_Sent. Протокол ведь не выдаёт DISind при переходе по p?DC. Этот переход можно рассматривать как неточность спецификации.

Были выявлены ошибки в CTesK.

3.6 Особенности тестирования ABRACADABRA

При тестировании протокола ABRACADABRA было применено тестирование таймаутов и подтверждений на повторные передачи.

4 Заключение

Данная работа показала применимость технологии автоматизированного тестирования UniTESK для тестирования реализаций протоколов с установлением соединения.

В рамках работы была разработана формальная спецификация протокола ABRACADABRA и разработан тестовый сценарий для проверки соответствия поведения реализации протокола спецификации.

Важной особенностью данной работы является то, что технология UniTESK позволяет автоматизировать тестирование таймаутов и повторных передачи, что является существенным препятствием для многих методов автоматизированного построения тестов из формальных спецификаций.

Сравнение с другими языками формального описания протоколов показало, что размер спецификации протокола ABRACADABRA на расширении языка Си приблизительно равен размеру спецификаций на языках Estelle и Lotos, то есть технология UniTESK позволяет достигать более высокого уровня автоматизации тестирования, нежели другие методы автоматической генерации тестов их формальных спецификаций при аналогичном размере формальной спецификации.

5 Список литературы

[1] Kenneth Turner. Using Formal Description Techniques. John Wiley and Sons Ltd., 1993. C. 431

[2] И. Б. Бурдонов, А. С. Косачев, В. В. Кулямин, А. К. Петренко. Архитектура тестового набора UniTesK. Труды FME'2002, Копенгаген, Дания. С.12

6 Приложения

В даном приложении включены файлы формальной спецификации (abra.seh и abra.sec) и тестовые сценарии (abra_scenario.sec)

6.1 Заголовочный файл формальной спецификации

Декларации, необходимые для формальной спецификации и тестового сценария, были собраны в заголовочный файл abra.seh

```
#ifndef ABRA_SEH__INCLUDED
#define ABRA_SEH__INCLUDED
#include <atl/list.h>
#include <atl/unit.h>
struct _DT_PDU {
    int ns;
    List * data;
};
#ifdef __SEC__
#pragma SEC file
  specification typedef struct _DT_PDU DT_PDU;
#else
  typedef struct _DT_PDU DT_PDU;
  extern const Type type_DT_PDU;
#endif
struct _ACK_PDU {
    int nr;
};
#ifdef __SEC__
  #pragma SEC file
  specification typedef struct _ACK_PDU ACK_PDU;
#else
  typedef struct _ACK_PDU ACK_PDU;
  extern const Type type_ACK_PDU;
#endif
typedef enum {
    CLOSED,
    OUT_CALL,
    IN_CALL,
    DATA_TRANSFER,
    DR_SENT
} state_name_t;
const char * get_state_name(int state_id);
const char * get_current_state_name();
extern state_name_t current_state;
extern int vs;
extern int vr;
extern int max_retries_param;
extern int cnt;
extern bool should_send_CR;
extern bool should_send_CC;
extern bool should_send_DR;
```

```
extern bool should_send_DC;
extern bool should_send_CONcnf;
extern bool should_send_CONind;
extern bool should_send_DISind;
extern bool should_send_DATind;
extern bool should_send_ACK;
extern bool should_send_DT;
extern bool sending;
List * get_data_sent();
void set_data_sent(List * value);
List * get_data_received();
void set_data_received(List * value);
bool should_not_send_anything(void);
specification void CONreq_spec(void);
specification void CONrsp_spec(void);
reaction Unit * CONind_spec(void);
reaction Unit * CONcnf_spec(void);
specification void DISreq_spec(void);
reaction Unit * DISind_spec(void);
specification void DATreq_spec(List * bytes);
reaction List * DATind_spec(void);
specification void receive_CR_spec(void);
reaction Unit * send_CR_spec(void);
specification void receive_CC_spec(void);
reaction Unit * send_CC_spec(void);
specification void receive_DR_spec(void);
reaction Unit * send_DR_spec(void);
specification void receive_DC_spec(void);
reaction Unit * send_DC_spec(void);
specification void receive_DT_spec(DT_PDU * dt);
reaction DT_PDU * send_DT_spec(void);
specification void receive_ACK_spec(ACK_PDU * ack);
reaction ACK_PDU * send_ACK_spec(void);
mediator CONreq_media for specification void CONreq_spec(void);
mediator CONrsp_media for specification void CONrsp_spec(void);
mediator CONind_media for reaction Unit * CONind_spec(void);
mediator CONcnf_media for reaction Unit * CONcnf_spec(void);
mediator DISreq_media for specification void DISreq_spec(void);
mediator DISind_media for reaction Unit * DISind_spec(void);
mediator DATreq_media for specification void DATreq_spec(List * bytes);
mediator DATind_media for reaction List * DATind_spec(void);
mediator receive_CR_media for specification void receive_CR_spec(void);
mediator send_CR_media for reaction Unit * send_CR_spec(void);
mediator receive_CC_media for specification void receive_CC_spec(void);
mediator send_CC_media for reaction Unit * send_CC_spec(void);
mediator receive_DR_media for specification void receive_DR_spec(void);
mediator send_DR_media for reaction Unit * send_DR_spec(void);
mediator receive_DC_media for specification void receive_DC_spec(void);
mediator send_DC_media for reaction Unit * send_DC_spec(void);
mediator receive_DT_media for specification void receive_DT_spec(DT_PDU * dt);
```

```
mediator send_DT_media for reaction DT_PDU * send_DT_spec(void);
mediator receive_ACK_media for specification void receive_ACK_spec(ACK_PDU * ack);
mediator send_ACK_media for reaction ACK_PDU * send_ACK_spec(void);
Object * abra_save_state();
void abra_restore_state(Object * obj);
#endif                                  // ABRA_SEH__INCLUDED
```

6.2 Формальная спецификация протокола ABRACADABRA

```
#include "abra.seh"
#include <stdio.h>
char * state_names[] = {
    "CLOSED",
    "OUT_CALL",
    "IN_CALL",
    "DATA_TRANSFER",
    "DR_SENT"
};
const char * get_state_name(int state_id) { return state_names[state_id]; }
const char * get_current_state_name() { return get_state_name(current_state);
}
specification typedef struct _DT_PDU DT_PDU = {};
specification typedef struct _ACK_PDU ACK_PDU = {};
state_name_t current_state = CLOSED;
int vs = 0;
int vr = 0;
int cnt = 0;
int max_retries_param = 3;
bool sending = false;
bool should_send_CR = false;
bool should_send_CC = false;
bool should_send_DR = false;
bool should_send_DC = false;
bool should_send_CONcnf = false;
bool should_send_CONind = false;
bool should_send_DISind = false;
bool should_send_DATind = false;
bool should_send_ACK = false;
bool should_send_DT = false;
bool should_not_send_anything()
{
    return !sending
          && !should_send_CR&& !should_send_CC&& !should_send_DR&&
!should_send_DC
          && !should_send_CONcnf&& !should_send_CONind&& !should_send_DISind
          && !should_send_DATind&& !should_send_ACK&& !should_send_DT;
}
List * _data_sent = NULL;
List * _data_received = NULL;
List * get_data_sent()
```

```
{
    return _data_sent;
}
void set_data_sent(List * value)
{
    _data_sent = value;
}
List * get_data_received()
{
    return _data_received;
}
void set_data_received(List * value)
{
    _data_received = value;
}
reaction Unit * CONind_spec(void)
{
    pre
    {
        return should_send_CONind;
    }
    post
    {
           return current_state == IN_CALL
             && !should_send_CONind
             && cnt == 0;
    }
}
specification void CONreq_spec(void)
{
    pre
    {
        return current_state == CLOSED || current_state == IN_CALL;
    }
    coverage C
     {
           if (current_state == CLOSED) return {out_call, "Outcoming call"};
           else return { simultan_call, "Simultanious call by both nodes" };
     }
     post
    {
        if (@current_state == CLOSED)
        {
            return current_state == OUT_CALL
                && cnt == max_retries_param
                && should_send_CR;
        }
        else                             // IN_CALL
        {
            return current_state == DATA_TRANSFER&& cnt == 0&& should_send_CC
```

```
                && should_send_CONcnf;
        }
    }
}
specification void CONrsp_spec(void)
{
    pre
    {
        return current_state == IN_CALL;
    }
    post
    {
        return current_state == DATA_TRANSFER&& cnt == 0&& should_send_CC;
    }
}
reaction Unit * CONcnf_spec(void)
{
    pre
    {
        return should_send_CONcnf && current_state == DATA_TRANSFER;
    }
    post
    {
          return current_state == @current_state&& !should_send_CONcnf&& cnt
== 0;
    }
}
specification void DISreq_spec(void)
{
    pre
    {
          bool res = current_state == IN_CALL || current_state == OUT_CALL
|| current_state == DATA_TRANSFER;
          fprintf(stderr, "SPEC: DISreq_spec: current_state %s, precondition
%d\n", state_names[current_state], res);
        return res;
    }
    coverage C
    {
          if (current_state == IN_CALL) return { in_break, "Cancel incoming
call" };
          else if (current_state == OUT_CALL) return { out_break, "Cancel
outcoming call" };
          else return { disreq_sent, "Finishing connection" };
    }
    post
    {
        return current_state == DR_SENT
                && cnt == max_retries_param
                && should_send_DR;
    }
```

```
}
reaction Unit * DISind_spec(void)
{
    pre
    {
        if (!should_send_DISind)
        {
            return false;
        }

        return (current_state == DR_SENT || current_state == CLOSED)
            || (  (current_state == OUT_CALL || current_state ==
DATA_TRANSFER)&& cnt == 0);
    }
    post
    {
        if (@current_state == DR_SENT || @current_state == CLOSED)
        {
            return !should_send_DISind
                && current_state == @current_state;
        }
        else if (@current_state == DATA_TRANSFER)
        {
            return current_state == DR_SENT&& should_send_DR&& cnt ==
max_retries_param
                && !should_send_DISind;
        }
        else
        {
            return current_state == CLOSED&& should_not_send_anything();
        }
    }
}
specification void DATreq_spec(List * bytes)
{
    pre
    {
        return current_state == DATA_TRANSFER && !sending;
    }
    post
    {
        return current_state == @current_state&& sending&& cnt ==
max_retries_param
            && should_send_DT&& vs == @vs + 1;
    }
}
reaction List * DATind_spec(void)
{
    pre
    {
        return should_send_DATind && current_state == DATA_TRANSFER;
```

```
    }
    post
    {
        return current_state == @current_state&& equals(DATind_spec,
get_data_received())
                && !should_send_DATind;
    }
}
specification void receive_CR_spec(void)
{
    pre
    {
        return current_state == CLOSED || current_state == OUT_CALL;
    }
     coverage C
     {
          if (current_state == CLOSED) return {in_call, "Incoming call"};
          else return { simultan_call, "Simultanious call by both nodes" };
     }
    post
    {
        if (@current_state == CLOSED)
        {
            return current_state == IN_CALL&& cnt == 0&& should_send_CONind;
        }
        else
        {
            return current_state == DATA_TRANSFER&& cnt == 0&& !should_send_CR
                && should_send_CC&& should_send_CONcnf;
        }
    }
}
reaction Unit * send_CR_spec(void)
{
    pre
    {
        return cnt > 0 && should_send_CR && current_state == OUT_CALL;
    }
    post
    {
        return cnt == @cnt - 1&& (cnt > 0)? should_send_CR : (!should_send_CR
&& should_send_DISind)
            && current_state == OUT_CALL;
    }
}
specification void receive_CC_spec(void)
{
    pre
    {
        return current_state == OUT_CALL;
    }
```

```
    coverage C
    {
        if (cnt == 0) return { last_CC, "CC sent to last CR" };
          else if (cnt == max_retries_param-1) return { first_CC, "CC sent to
the first CR" };
          else return { intermediate_CC, "CC sent to an intermediate CR" };
    }
    post
    {
      return current_state == DATA_TRANSFER&& cnt == 0&& should_send_CONcnf;
    }
}
reaction Unit * send_CC_spec(void)
{
    pre
    {
        return should_send_CC && current_state == DATA_TRANSFER;
    }
    post
    {
        return current_state == @current_state && !should_send_CC;
    }
}
specification void receive_DR_spec(void)
{
    pre
    {
        return current_state == IN_CALL || current_state == OUT_CALL
|| current_state == DATA_TRANSFER;
    }
    coverage C
     {
          if (current_state == IN_CALL) return { in_break, "Breaking incoming
call" };
          else if (current_state == OUT_CALL) return { out_break, "Breaking
outcoming call" };
          else return { disreq_sent, "Finishing connection" };
     }
     post
    {
        return current_state == CLOSED&& cnt == 0&& should_send_DISind&&
should_send_DC;
    }
}
reaction Unit * send_DR_spec(void)
{
    pre
    {
        return should_send_DR && (   (current_state == DR_SENT && cnt > 0)
                    || (current_state == DATA_TRANSFER && cnt == 0));
    }
```

```
    post
    {
           if (@current_state == DR_SENT)
           {
                 if (@cnt > 1)
                 {
                        return cnt == @cnt - 1&& should_send_DR &&
current_state == @current_state;
                 }
                 else
                 {
                       return should_not_send_anything()&& current_state ==
CLOSED;
                 }
           }
           else
           {
                 return current_state == DR_SENT && cnt == max_retries_param -
1
                       && should_send_DR;
           }
    }
}
specification void receive_DC_spec(void)
{
    pre
    {
        return current_state == DR_SENT;
    }
    coverage C
    {
        if (cnt == 0) return { last_DC, "DC sent to last DR" };
          else if (cnt == max_retries_param-1) return { first_DC, "DC sent to
the first DR" };
          else return { intermediate_DC, "DC sent to an intermediate DR" };
    }
    post
    {
        return current_state == CLOSED&& should_send_DR == false&& cnt == 0;
    }
}
reaction Unit * send_DC_spec(void)
{
    pre
    {
        return should_send_DC && current_state == CLOSED;
    }
    post
    {
       return current_state == @current_state&& !should_send_DC;
    }
```

```
}
specification void receive_DT_spec(DT_PDU * dt)
{
    pre
    {
        return current_state == DATA_TRANSFER;
    }
    coverage seq_num
    {
        if (dt->ns == vr)
        {
            return {new_dt, "New data transfer message"};
        }
        else if (dt->ns == vr - 1)
        {
            return {resent_dt, "A data transfer message received once again"};
          }
        else
        {
            return {wrong_dt, "A data transfer message with incorrect sequence
number"};
        }
    }
    post
    {
        if (coverage(seq_num) == new_dt)
        {
            return vr == @vr + 1 && current_state == @current_state
                && should_send_DATind && should_send_ACK&& cnt == 0;
        }
        else if (coverage(seq_num) == resent_dt)
        {
            return  current_state == @current_state&& should_send_ACK&& cnt ==
0;
       }
        else
        {
            return  current_state == @current_state&& cnt == 0;
        }
    }
}
reaction DT_PDU * send_DT_spec(void)
{
    pre
    {
        return cnt > 0 && should_send_DT && current_state == DATA_TRANSFER;
    }
    post
    {
        return cnt == @cnt - 1
```

```
            && equals(get_data_sent(), send_DT_spec->data) && (cnt > 0 &&
sending)? should_send_DT : (!should_send_DT && should_send_DISind &&
should_send_DR) && current_state == @current_state;
    }
}
specification void receive_ACK_spec(ACK_PDU * ack)
{
    pre
    {
        return current_state == DATA_TRANSFER;
    }
    coverage C
    {
        if (cnt == 0) return { last_AK, "AK sent to last DT" };
          else if (cnt == max_retries_param-1) return { first_AK, "AK sent to
the first DT" };
          else return { intermediate_CC, "AK sent to an intermediate DT" };
    }
    post
    {
        if (ack->nr > vs)
        {
            return current_state == DR_SENT
              && should_send_DR && should_send_DISind
              && cnt == max_retries_param;
        }
        else
        {
            return current_state == @current_state&& !sending&& cnt == 0&&
should_not_send_anything();
        }
    }
}
reaction ACK_PDU * send_ACK_spec(void)
{
    pre
    {
        return should_send_ACK && current_state == DATA_TRANSFER;
    }
    post
    {
        return current_state == @current_state && (!should_send_ACK);
    }
}
struct model_state
{
    state_name_t current_state;
    int vs;
    int vr;

    int max_retries_param;
```

```
    int cnt;
    bool should_send_CR;
    bool should_send_CC;
    bool should_send_DR;
    bool should_send_DC;
    bool should_send_CONcnf;
    bool should_send_CONind;
    bool should_send_DISind;
    bool should_send_DATind;
    bool should_send_ACK;
    bool should_send_DT;
    bool sending;
    List * data_sent;
    List * data_received;
};
specification typedef struct model_state ModelState = {};
Object * abra_save_state()
{
    return create(&type_ModelState,
                      current_state,
                      vs,
                      vr,
                      max_retries_param,
                      cnt,
                      should_send_CR,
                      should_send_CC,
                      should_send_DR,
                      should_send_DC,
                      should_send_CONcnf,
                      should_send_CONind,
                      should_send_DISind,
                      should_send_DATind,
                      should_send_ACK,
                      should_send_DT,
                      sending,
                      clone(get_data_sent()),
                      clone(get_data_received())
          );
}
void abra_restore_state(Object * obj)
{
    ModelState * mstate = (ModelState*)obj;
    current_state = mstate->current_state;
    vs = mstate->vs;
    vr = mstate->vr;
    max_retries_param = mstate->max_retries_param;
    cnt = mstate->cnt;
    should_send_CR = mstate->should_send_CR;
    should_send_CC = mstate->should_send_CC;
    should_send_DR = mstate->should_send_DR;
```

```
    should_send_DC = mstate->should_send_DC;
    should_send_CONcnf = mstate->should_send_CONcnf;
    should_send_CONind = mstate->should_send_CONind;
    should_send_DISind = mstate->should_send_DISind;
    should_send_DATind = mstate->should_send_DATind;
    should_send_ACK = mstate->should_send_ACK;
    should_send_DT = mstate->should_send_DT;
    sending = mstate->sending;
    set_data_sent(mstate->data_sent);
    set_data_received(mstate->data_received);
}
```

6.3 Тестовый сценарий для тестирования соответствия спецификации протокола ABRACADABRA

Файл abra_scenario.sec:

```
#include "abra.seh"
#include "abra_lower_tester.seh"
#include "abra_upper_tester.seh"
#include <atl/char.h>
#include "service.h"
#include <atl/integer.h>
#include <ts/timemark.h>
#include <windows.h>
unsigned int timeout_delay_millis = 0;
ChannelID scenario_channel = -1;
bool init_scenario(int argc, char ** argv)
{
    set_mediator_CONreq_spec(CONreq_media);
    set_mediator_CONrsp_spec(CONrsp_media);
    set_mediator_CONcnf_spec(CONcnf_media);
    set_mediator_CONind_spec(CONind_media);
    set_mediator_DISreq_spec(DISreq_media);
    set_mediator_DISind_spec(DISind_media);
    set_mediator_DATreq_spec(DATreq_media);
    set_mediator_DATind_spec(DATind_media);
    set_mediator_receive_CR_spec(receive_CR_media);
    set_mediator_send_CR_spec(send_CR_media);
    set_mediator_receive_CC_spec(receive_CC_media);
    set_mediator_send_CC_spec(send_CC_media);
    set_mediator_receive_DR_spec(receive_DR_media);
    set_mediator_send_DR_spec(send_DR_media);
    set_mediator_receive_DC_spec(receive_DC_media);
    set_mediator_send_DC_spec(send_DC_media);
    set_mediator_receive_DT_spec(receive_DT_media);
    set_mediator_send_DT_spec(send_DT_media);
    set_mediator_receive_ACK_spec(receive_ACK_media);
    set_mediator_send_ACK_spec(send_ACK_media);
    scenario_channel = getChannelID();
    setStimulusChannel(scenario_channel);
```

```
    fprintf(stderr,  "SCENARIO: scenario_channel %d\n", scenario_channel);
    fprintf(stderr,  "SCENARIO: unique channel %d\n", UniqueChannel);
    setWTime(2);
    if (SE_SUCCESS != init_service(client_event_handler, 0)) return false;
    if (SE_SUCCESS != set_service_option(ABRA_TRIES, max_retries_param))
return false;
    if (SE_SUCCESS != get_service_option(ABRA_CONNECT_TIMEOUT,
&timeout_delay_millis)) return false;
}
void finish_scenario()
{
    finish_service();
}
Object * simple_scenario_state()
{
    return create_Integer(current_state);
}
scenario bool outcall_scen()
{
    if (pre_CONreq_spec() && current_state == CLOSED)
    {
        iterate (int receive_CC = -1; receive_CC <= max_retries_param-1;
receive_CC ++; )
        {
            fprintf(stderr, "Sending CONreq\n");
            CONreq_spec();
            if (receive_CC >= 0)
            {
                Sleep(timeout_delay_millis*receive_CC +
timeout_delay_millis/2);
                fprintf(stderr, "Send CC to the SUT\n");
                // receive_CC_spec();
                registerStimulusWithTimeInterval(get_link_channel(),
NULL, receive_CC_spec,
                    createTimeInterval(minTimeMark, maxTimeMark));
                abra_packet_stimulus(0, ABRA_PACKET_CC, 0, NULL);
            }
        }
    }
}
scenario bool outcall_dr_scen()
{
    if (pre_CONreq_spec() && current_state == CLOSED)
    {
        iterate (int receive_DR = 0; receive_DR <= max_retries_param-2;
receive_DR ++; )
        {
            fprintf(stderr, "Sending CONreq\n");
            CONreq_spec();
            Sleep(timeout_delay_millis*receive_DR +
timeout_delay_millis/2);
            fprintf(stderr, "Send DR to the SUT\n");
```

```
                // receive_DR_spec();
                registerStimulusWithTimeInterval(get_link_channel(), NULL,
receive_DR_spec,
                        createTimeInterval(minTimeMark, maxTimeMark));
                abra_packet_stimulus(0, ABRA_PACKET_DR, 0, NULL);
            }
    }
}
scenario bool incall_scen()
{
    if (pre_receive_CR_spec())
    {
            fprintf(stderr, "Sending CR\n");
            setStimulusChannel(UniqueChannel);
            receive_CR_spec();
            setStimulusChannel(scenario_channel);
    }
}
scenario bool incall_conreq()
{
    if (pre_CONreq_spec() && current_state == IN_CALL)
    {
            fprintf(stderr, "Sending CONreq\n");
            CONreq_spec();
    }
}
scenario bool incall_accept()
{
    if (pre_CONrsp_spec())
    {
            fprintf(stderr, "Sending CONrsp\n");
            CONrsp_spec();
    }
}
scenario bool outcall_with_incall_scen()
{
    if (pre_CONreq_spec() && current_state == CLOSED)
    {
            iterate (int receive_CR = -1; receive_CR < max_retries_param;
receive_CR ++; )
            {
                fprintf(stderr, "Sending CONreq\n");
                CONreq_spec();
                if (receive_CR >= 0)
                {
                        Sleep(timeout_delay_millis*receive_CR +
timeout_delay_millis/2);
                        fprintf(stderr, "Send CR to the SUT\n");
                        registerStimulusWithTimeInterval(get_link_channel(),
NULL, receive_CR_spec,
                                createTimeInterval(minTimeMark, maxTimeMark));
```

```
                    abra_packet_stimulus(0, ABRA_PACKET_CR, 0, NULL);
                }
            }
    }
}
scenario bool cancel_outcall_scen()
{
    if (pre_CONreq_spec() && current_state == CLOSED)
    {
            int res;
            fprintf(stderr, "Sending CONreq\n");

            CONreq_spec();
            Sleep(timeout_delay_millis/2);
            fprintf(stderr, "Cancel outcall\n");
            registerStimulusWithTimeInterval(scenario_channel, NULL,
DISreq_spec,
                    createTimeInterval(minTimeMark, maxTimeMark));
            res = abra_disconnect();
        if (res != SE_SUCCESS)
        {
                return false;
            }
    }
}
scenario bool user_disconnect_scen()
{
    if (pre_DISreq_spec())
    {
            iterate (int receive_DC = -1; receive_DC < max_retries_param-1;
receive_DC ++; )
            {
                fprintf(stderr, "Sending DISreq\n");
                DISreq_spec();
                if (receive_DC >= 0)
                {
                    Sleep(timeout_delay_millis*receive_DC +
timeout_delay_millis/2);
                    fprintf(stderr, "Send DC to the SUT\n");
                    abra_packet_stimulus(0, ABRA_PACKET_DC, 0, NULL);
                    registerStimulusWithTimeInterval(get_link_channel(),
NULL, receive_DC_spec,
                        createTimeInterval(minTimeMark, maxTimeMark));
                }
            }
    }
}
scenario bool user_disconnect_with_dr_scen()
{
    if (pre_DISreq_spec())
    {
```

```
          iterate (int receive_DR = 0; receive_DR < max_retries_param-2;
receive_DR ++; )
          {
                fprintf(stderr, "Sending DISreq\n");
                DISreq_spec();
                Sleep(timeout_delay_millis*receive_DR +
timeout_delay_millis/2);
                fprintf(stderr, "Send DR to the SUT\n");
                registerStimulusWithTimeInterval(get_link_channel(), NULL,
receive_DR_spec,
                      createTimeInterval(minTimeMark, maxTimeMark));
                abra_packet_stimulus(0, ABRA_PACKET_DR, 0, NULL);
          }
     }
}
scenario bool data_transfer_send_scen()
{
     List * bytes = create_List(NULL);
     if (pre_DATreq_spec(bytes))
     {
          iterate (int receive_AK = -1; receive_AK < max_retries_param;
receive_AK ++; )
          {
                fprintf(stderr, "Sending DATreq\n");
                DATreq_spec(bytes);
                if (receive_AK >= 0)
                {
                      int ak_vs = vs+1;
                      Sleep(timeout_delay_millis*receive_AK +
timeout_delay_millis/2);
                      fprintf(stderr, "Send AK to the SUT\n");
                      abra_packet_stimulus(ak_vs, ABRA_PACKET_AK, 0, NULL);
                      registerStimulusWithTimeInterval(get_link_channel(),
NULL, receive_ACK_spec,
                            createTimeInterval(minTimeMark, maxTimeMark),
                            create(&type_ACK_PDU, ak_vs),
create(&type_ACK_PDU, ak_vs));
                }
          }
     }
}
scenario bool disconnect_request_scen()
{
     if (pre_receive_DR_spec() && (current_state == DATA_TRANSFER ||
current_state == IN_CALL))
     {
          fprintf(stderr, "Sending DR\n");
          setStimulusChannel(UniqueChannel);
          receive_DR_spec();
          setStimulusChannel(scenario_channel);
     }
}
```

```
static char data_str[] = "Hello, world!";
List * get_test_data()
{
    static List * test_data = NULL;
    if (test_data == NULL)
    {
        unsigned int i;
        test_data = create_List(NULL);
        for (i = 0; i < sizeof(data_str); i++)
        {
            append_List(test_data, create_Char(data_str[ i ]));
        }
    }
    return test_data;
}
DT_PDU * create_test_DT(int num)
{
    return create(&type_DT_PDU, num, get_test_data());
}
enum { DT_NEW_MESSAGE, DT_RESENT, DT_INCORRECT_NUMBER };
scenario bool data_transfer_recieve_scen()
{
     iterate (int action = DT_NEW_MESSAGE; action <= DT_INCORRECT_NUMBER;
action ++; )
     {
          int seq_num;
          DT_PDU * dt = NULL;
          switch(action)
          {
          case DT_NEW_MESSAGE: seq_num = vr; break;
          case DT_RESENT: seq_num = vr-1; break;
          default: seq_num = vr+100;
          }
          dt = create_test_DT(seq_num);
          if (pre_receive_DT_spec(dt))
          {
                fprintf(stderr, "Sending Data\n");
                setStimulusChannel(UniqueChannel);
                receive_DT_spec(dt);
                setStimulusChannel(scenario_channel);
          }
     }
}
bool simple_scenario_stationary_state()
{
     bool res = should_not_send_anything();
     if (res)
          fprintf(stderr, "SCENARIO: stationary state\n");
     else
          fprintf(stderr, "SCENARIO: NON stationary state\n");
     fprintf(stderr, "SCENARIO: current state %s\n", get_current_state_name());
```

```
    return res;
}
scenario dfsm simple_scenario =
{
    .init = init_scenario,
          .finish = finish_scenario,
          .getState = simple_scenario_state,
          .saveModelState = abra_save_state,
          .restoreModelState = abra_restore_state,
          .isStationaryState = simple_scenario_stationary_state,

          .actions = { outcall_scen, outcall_with_incall_scen,
outcall_dr_scen, cancel_outcall_scen,
                       data_transfer_send_scen, data_transfer_recieve_scen,
                       incall_scen, incall_accept, incall_conreq,
                       user_disconnect_scen, disconnect_request_scen,
user_disconnect_with_dr_scen,
                             NULL},
};
int main(int argc, char **argv)
{
    bool res = simple_scenario(argc, argv);
    return (res)?0:1;
}
```