Интернет-журнал «Науковедение» ISSN 2223-5167 http://naukovedenie.ru/

Том 8, №5 (2016) http://naukovedenie.ru/index.php?p=vol8-5

URL статьи: http://naukovedenie.ru/PDF/50TVN516.pdf

Статья опубликована 31.10.2016

Ссылка для цитирования этой статьи:
Мьё Тхет Наунг, Девятков В.В. Формальная проверка правильности сетевого взаимодействия агентов, специфицируемых на языке процессных выражений // Интернет-журнал «НАУКОВЕДЕНИЕ» Том 8, №5 (2016) http://naukovedenie.ru/PDF/50TVN516.pdf (доступ свободный). Загл. с экрана. Яз. рус., англ.

УДК 004.413

Мьё Тхет Наунг
ФГБОУ ВПО «Московский государственный технический университет им. Н.Э. Баумана», Россия, Москва
Аспирант кафедры «Информационные системы и телекоммуникации»
E-mail: komyothetnaung@gmail.com

Девятков Владимир Валентинович
ФГБОУ ВПО «Московский государственный технический университет им. Н.Э. Баумана», Россия, Москва
Зав. кафедры «Информационные системы и телекоммуникации»
Доктор технических наук, профессор
E-mail: deviatkov@ bmstu.ru

Формальная проверка правильности сетевого взаимодействия агентов, специфицируемых на языке процессных выражений

Аннотация. Настоящая статья посвящена развитию методики формальной проверки правильности взаимодействия агентов, моделирующих сетевое взаимодействие. Поведение агентов специфицируется на языке процессных выражений. Взаимодействующие агенты широко применяются для моделирования сетевых взаимодействий, использующих различные протоколы. Автоматизацию проверки правильности спецификаций предлагается осуществлять логическими программами, получаемыми с помощью представленной в настоящей статье методики перехода от канонических процессных выражений, специфицирующих поведение агентов, а также требований правильности взаимодействия на языке модальной логики. В качестве языка логического программирования для проверки правильности сетевого взаимодействия используется Visual Prolog. Настоящая статья развивает методику, изложенную в предыдущих статьях авторов. Основное отличие изложенного в статье материала от уже опубликованных результатов состоит в использовании не графового описания поведения агентов, а канонического символического, ограничивающего выразительность описания, но облегчающего переход от него к логической программе и формулировку требований правильности на языке модальной логики. Методика формальной проверки правильности сетевого взаимодействия иллюстрируется простым примером логической программы, на одной из ранних версий языка Visual Prolog, позволяющей проверять требования обязательной реакции для случая двух взаимодействующих агентов.

Ключевые слова: агент; сетевое взаимодействие; модальная логика; проверка правильности; канонические процессные выражения; язык логического программирования.

1. Введение

Для описания поведения агентов и их взаимодействия могут использоваться различные языки. Так, например, в работе [1] использовались графы переходов. Одними из самых популярных языков, используемых для этих целей, является язык, который будем называть языком процессных выражений, являющийся вариантом языка алгебры (пи-исчисления) Роберта Милнера [2]. Процессные выражения предлагают достаточно естественный путь описания параллельных систем, состоящих из процессов, взаимодействующих по общим каналам. Однако главный вопрос, который мы хотели бы обсудить, – это какими должны быть процессные выражения, для того, чтобы с их помощью можно было однотипно описывать различные архитектуры взаимодействующих агентов с возможностью как можно более эффективной проверки различных требований к правильности взаимодействия по этим процессным выражениям.

Для такого обсуждения воспользуемся спецификацией на языке процессных выражений двух простых взаимодействующих агентов UAC и UAS, рассмотренных в работе [1] и унаследованных из работы [2]. Взаимодействие агентов осуществляется с помощью обмена сообщениями по 10 каналам, имена которых ackc, brpc, irpc, reqc, sakc, acks, brps, irps, reqs, saks (рис. 1).

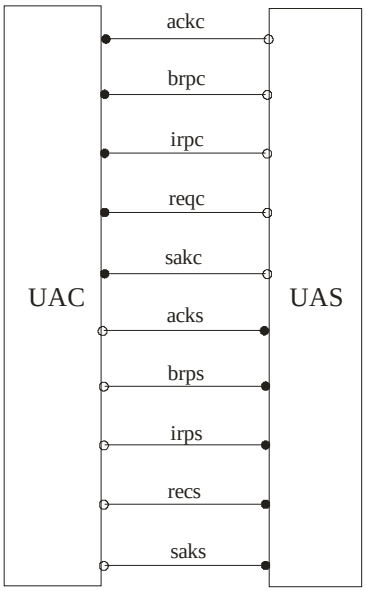

Рисунок 1. *Взаимодействующие агенты UAC и UAS (рисунок авторов)*

Каналы ackc, brpc, irpc, reqc, sakc являются входными для агента UAC и выходными для агента UAS, а каналы acks, brps, irps, reqs, saks наоборот − выходными для агента UAC и входными для агента UAS. Каждый из агентов может выполнять только шесть типов действий соответственно: invitec, invsuccc, invfailc, ackc, byec, byeRspc и invites, invsuccs, invfails, acks, byes, byeRsps. Если какое-либо сообщение *a* берется из канала с именем *ch*, то оно является восприятием и записывается как *ch*?*a*, если же оно выдается в канал *c*, то оно является реакцией и записывается как *ch*!*a*.

Процессные выражения, описывающие поведение агентов UAC и UAS подробно представлены ниже. Описание начинается с выражения UA=UAC|| UAS, задающего параллельное поведение агентов. Описание каждого агента начинается с описания одноименного процесса верхнего уровня (UAC или UAS) указанием в левой части процессного выражения имени процесса, а в правой после знака равенства - имени канала и действия, связанного с этим каналом, а затем подпроцесса, вызываемого после совершения действия. В такой же форме описываются подпроцессы процессов, подпроцессы

подпроцессов и т.д. использование каналов и действий может быть недетерминированным, что указывается знаком. Взаимодействие агентов начинается с выполнения процессов верхнего уровня. Какой из агентов является инициатором взаимодействия оговаривается особо.

UA=UAC||UAS

UAC = reqc!invite.Inviting	Процесс UAC посылает в свой выходной канал reqc запрос !invite процессу UAS на разрешение обслуживания и вызывает процесс Inviting.
Inviting =	Процесс Inviting инициации связи с процессом UAS.
ackc!ack.0	Процесс Inviting посылает в свой выходной канал ackc сообщение !ack для дальнейшего обслуживания.
| irps?invFail.Ending	Процесс Inviting процесса UAC получает в своем входном канале irps сообщение ?invFail от процесса UAS об отказе обслуживания по запросу !invite и переходит к выполнению процесса Ending окончания связи с процессом UAS.
| irps?invSucc.sakc!ack.Confirming	Процесс Inviting процесса UAC получает в своем входном канале irps сообщение ?invSucc от процесса UAS о согласии на обслуживание сообщения !ack, после чего процесс UAC, выдает в канал sakc уведомление о получении согласия на обслуживание сообщения !ack и переходит к выполнению процесса Confirming.
| reqs?bye.brpc!byeRsp.Ending	Процесс Inviting процесса UAC получает в своем входном канале reqs сообщение ?bye от процесса UAS о прекращении с ним связи, выдает в ответ в канал brpc прощальное сообщении !byeRsp процессу UAS и переходит к процессу Ending окончания связи с процессом UAS.
| brps?byeRsp.0	Процесс Inviting процесса UAC получает в своем входном канале brps входное прощальное сообщение ?byeRsp от процесса UAS о прекращении с ним связи и сразу переходит к процессу 0 прекращения своей активности.
Confirming =	Процесс Confirming ожидающий конца обслуживания сообщения !ack процессом UAS.
irps?invFail.0	Процесс Confirming процесса UAC получает в своем входном канале irps сообщение ?invFail от процесса UAS о неправильном запросе !invite и прекращает свою активность.
| irps?invSucc.0	Процесс Confirming процесса UAC получает в своем входном канале irps сообщение ?invSucc от процесса UAS об успешном запросе !invite и прекращает свою активность.
| reqs?bye.brpc!byeRsp.	
brps?byeRsp.Ending	Процесс Confirming процесса UAC получает в своем входном канале reqs сообщение ?bye от процесса UAS о

	завершении обслуживания, выдает в ответ в канал brpc прощальное сообщение !byeRsp процессу UAS, получает от него в канале brps уведомление ?byeRsp о получении сообщения !byeRsp и переходит к процессу Ending окончания связи с процессом UAS.
reqc!bye.Byeing	Процесс Confirming процесса UAC выдает в свой выходной канал reqc выходное сообщение !bye процессу UAS о прекращении ожидания от него конца обслуживания и переходит к процессу Byeing.
Byeing =	Процесс Byeing извещения процессом UAC окончания связи с процессом UAS.
irps?invFail.0	Процесс Byeing процесса UAC получает в своем входном каналеirps сообщение ?invFail от процесса UAS о неправильном запросе !invite и прекращает свою активность.
irps?invSucc.0	Процесс Byeing процесса UAC получает в своем входном канале irps сообщение ?invSucc от процесса UAS об успешном запросе !invite и прекращает свою активност.
reqs?bye.brpc!byeRsp.0	Процесс Byeing процесса UAC получает в своем входном канале reqs сообщение ?bye от процесса UAS о прекращении с ним связи, выдает в ответ в канал brpc прощальное сообщение !byeRsp процессу UAS и прекращает свою активность.
brps?byeRsp.Ending	Процесс Byeing процесса UAC получает в своем входном канале brps входное сообщение ?bye от процесса UAS о прекращении с ним связи и переходит к процессу завершения работы процесса UAC.
Ending =	Процесс Ending завершения работы процесса UAC.
irps?invFail.0	Процесс Ending процесса UAC получает в своем входном канале irps сообщение ?invFail от процесса UAS о неправильном запросе !invite и прекращает свою активность.
irps?invSucc.0	Процесс Ending процесса UAC получает в своем входном каналеirps сообщение ?invSucc от процесса UAS об успешном запросе !invite и прекращает свою активность.
reqs?bye.brpc!byeRsp.0	Процесс Ending процесса UAC получает в своем входном канале reqs сообщение ?bye от процесса UAS о прекращении с ним связи, выдает в ответ в канал brpc прощальное сообщение !byeRsp процессу UAS и прекращает свою активность.
brps?byeRsp.0	Процесс Ending процесса UAC получает в своем входном канале brps входное прощальное сообщение ?byeRsp от процесса UAS и прекращает свою активность.

UAS = reqc?invite.Invited	Процесс UAS получает в своем входном канале и вызывает процесс Invited.
Invited =	Процесс Invited инициации связи с процессом UAC.
ackc?ack.0	Процесс Invited процесса UAS получает в своем входном канале ackc запрос на обслуживание ?ack от процесса UAC и переходит в неактивное состояние.
| reqc?bye.0	Процесс Invited процесса UAS получает в своем входном канале reqc сообщение ?bye о конце связи от процесса UAC и переходит в неактивное состояние.
| brpc?byeRsp.0	Процесс Invited процесса UAS получает в своем входном канале brpc прощальное сообщение ?byeRsp от процесса UAC и переходит в неактивное состояние.
| irps!invFail.Ended	Процесс Invited процесса UAS выдает в свой выходной канал irps сообщение!invFail о неправильном запросе !invite от процесса UAC и переходит к выполнению процесса Ended.
| irps!invSucc.Confirmed	Процесс Invited процесса UAS выдает в свой выходной канал irps сообщение !invSucc о правильном запросе !invite от процесса UAC и переходит к выполнению процесса Confirmed.
Confirmed =	Процесс Confirmed подтверждения связи с процессом UAC.
ackc?acked.acks!acked.0	Процесс Confirmed процесса UAS получает в своем входном канале ackc запрос на обслуживание ?ack от процесса UAC, выдает ему сообщение на согласие обслуживания !acked и переходит в неактивное состояние.
| reqc?bye.brps!byeRsp.0	Процесс Confirmed процесса UAS получает в своем входном канале reqc сообщение о конце связи ?bye от процесса UAC, выдает в свой выходной канал brps сообщение !byeRsp о прекращении с ним связи и переходит в неактивное состояние.
| brpc?byeRsp.Ended	Процесс Confirmed процесса UAS получает в своем входном канале brpc прощальное сообщение ?byeRsp от процесса UAC и переходит к выполнению процесса Ended.
Byed =	Процесс Byed извещения процесса UAC об окончании с ним связи процессом UAS.
ackc?ack.acks!acked.0	Процесс Byed процесса UAS получает в своем входном канале ackc запрос на обслуживание ?ack от процесса UAC, выдает ему в канал acks сообщение на согласие обслуживания !acked переходит в неактивное состояние.
| reqc?bye.brps!byeRsp.0	Процесс Byed процесса UAS получает в своем входном канале reqc сообщение о конце связи ?bye от процесса UAC, выдает в свой выходной канал brps сообщение !byeRsp о прекращении с ним связи и переходит в неактивное состояние.

| brpc?byeRsp.Ended	Процесс Byed процесса UAS получает в своем входном канале brpc прощальное сообщение ?byeRsp от процесса UAC и переходит к выполнению процесса Ended.
Ended =	Процесс Ended завершения работы процесса UAS.
ackc?ack.acks!acked.0	Процесс Ended процесса UAS получает в своем входном канале ackc запрос на обслуживание ?ack от процесса UAC, выдает ему в канал acks сообщение на согласие обслуживания !acked и переходит в неактивное состояние.
| reqc?bye.brps!byeRsp.0	Процесс Ended процесса UAS получает в своем входном канале reqc сообщение о конце связи ?bye от процесса UAC, выдает в свой выходной канал brps сообщение !byeRsp о прекращении с ним связи и переходит в неактивное состояние.
| brpc?byeRsp.0	Процесс Ended процесса UAS получает в своем входном канале brpc прощальное сообщение ?byeRsp от процесса UAC и переходит в неактивное состояние.

2. Формулировка и проверка требований к правильности взаимодействия агентов

Во многих работах, например, работе [3] для формулировки требуемых свойств интерфейсов предлагается использовать широко применяемую в области программных систем предварительную формулировку этих свойств на языке временной модальной логики. Поскольку этот язык хорошо изучен и известен [4, 5], то в настоящей статье не будем уделять ему слишком много внимания. Обсудим недостатки приведенного описания на языке процессных выражений поведения агентов UAC и UAS с точки зрения возможности эффективных проверок на примере простого требования правильности, называемого свойством обязательной реакции [3]. В простейшем случае суть этого свойства для нашего примера с двумя агентами UAC и UAS состоит в том, что в ответ на любую реакцию, поступающую в выходной канал одного агента, являющийся входным для другого, последний должен получить эту реакцию в виде восприятия и среагировать на него по своему выходному каналу, являющуюся входным для первого агента. Последний должен воспринять эту реакцию. Будем использовать введённые выше обозначения *chS* входных для агента С и выходных для агента UAS каналов и обозначения *chC* каналов входных для агента UAS и выходных для агента UAC. Тогда на языке модальной логики формулировка некоторого свойства обязательной реакции может выглядеть следующим образом:

$$\{(P^1_{uac} = chC\,!\,a_1) \supset \Diamond[(P^1_{uas} = chC\,?\,a_1) \supset \Diamond((P^2_{uas} = chS\,!\,a_2) \supset \Diamond(P^2_{uac} = chS\,?\,a_2))]\}$$

Здесь

$$P^1_{uac} = chC\,!\,a_1,\ P^1_{uas} = chC\,?\,a_1,\ P^2_{uas} = chS\,!\,a_2,\ P^2_{uac} = chS\,?\,a_2$$

- предикаты с предикатным символом “=”, представленные в инфиксной форме. Значение этих предикатов истинно, когда выполняется процесс, именуемый в левой части равенств, выполнение которого состоит в помещении в канал, именуемый в правой части сообщения, следующего после имени канала. Имена процессов, каналов и действий в этих предикатах являются переменными.

Формулировка любого свойства взаимодействия агентов на языке модальной логики – это по существу утверждение, которое требует определенной процедуры доказательства. В

частности, свойство обязательной реакции, сформулированное в разделе 2 предполагает, что поведение агентов UAC и UAS должно обладать определенным порядком выполнения процессов. Это означает, что в описании поведения агентов на языке процессных выражений для любого процессного выражения $P_{uac}^{1} = chC!a_1$ агента UAC должно найтись процессное выражение процесса $P_{uas}^{1} = chC?a_1$ агента UAS, а для любого процессного выражения $P_{uas}^{2} = chS!a_2$ агента UAS процессное выражение $P_{uac}^{2} = chS?a_2$ агента UAC, причем выполнение процессов P_{uac}^{1}, P_{uas}^{1}, P_{uas}^{2}, P_{uac}^{2} осуществляется в порядке слева направо. Приведенное свойство обязательной реакции не единственное. В принципе, этих свойств может быть сколь угодно много, например, в зависимости от структуры восприятий и реакций, процессов, осуществляющих их выдачу и получение, и, может быть, каких- либо дополнительных условий.

Глядя на приведенное выше описание поведения процессов UAC и UAS, с точки зрения возможности организации формальной проверки по этому описанию правильности взаимодействия можно выделить следующие недостатки этого описания.

1. Имена подпроцессов вводятся, исходя из субъективных соображений функционального разбиения процессов UAC и UAS на подпроцессы.
2. Подпроцессы могут состоять из подпроцессов, разделяемых знаком | и никак не поименованных.
3. Нити действий подпроцессов могут иметь различную длину.
4. Имена каналов имеют обозначения, которые, в случае числа агентов большего двух, не позволяют определить, каким агентам они смежны.
5. Число уровней процессов ограничено двумя.
6. Функциональность процессов ограничена восприятиями и реакциями, помещаемыми в каналы.
7. Нет никакой информации о том, в какие каналы в ответ на реакцию должно поступать восприятие и наоборот.

Указанные недостатки описания взаимодействия агентов даже для проверки свойства обязательной реакции требуют выделения всех взаимодействующих подпроцессов и каналов по которым это взаимодействие осуществляется. Это может быть осуществлено преобразованием процессных выражений к некоторому каноническому виду, в котором указанное выделение осуществлено. Для случая простого взаимодействия двух агентов UAC и UAS это преобразование представляется разрешимым, поскольку обозначения каналов однозначно определяют, какие из них являются входными или выходными для каждого процесса, а подпроцессы, имеющие длину нитей более 1, могут быть разбиты на подпроцессы с именами, удовлетворяющими формулировке свойства обязательной реакции.

В более сложных случаях взаимодействия агентов разрешимость преобразования далеко не очевидна. Поэтому на наш взгляд предпочтительнее идти по пути не преобразований, а по пути использования для описания поведения агентов канонических процессных выражений.

3. Канонические процессные выражения

В канонических процессных выражениях используются нити единично длины. Продемонстрируем использование канонических процессных выражений на следующем простейшем примере.

$$\begin{aligned} P &= P_a \,\|\, P_b\,, \\ P_a &= ch(P_a P_b)\,!a_1\, P_{a1}, \\ P_{a1} &= ch(P_b P_a)\,?a_2.0, \\ P_b &= ch(P_a P_b)\,?a_1\, P_{b1}, \\ P_{b1} &= ch(P_b P_a)\,!a_2.0 \end{aligned}$$

Здесь процессное выражение P задает два параллельных процесса P_a и P_b. Процессное выражение P_a описывает поведение одноименного агента P_a, а процессное выражение P_b описывает поведение агента P_b. Процессы P_{a1} и P_{b1}, являются подпроцессами соответственно процессов P_a и P_b. Процессы и соответственно агенты P_a и P_b взаимодействуют по двум каналам $ch(P_a P_b)$ и $ch(P_b P_a)$. Канал $ch(P_a P_b)$ является выходным для процесса P_a и входным для процесса P_b. Канал $ch(P_b P_a)$ является выходным для процесса P_b и входным для процесса P_b. Свойство обязательной реакции взаимодействия для процессов P_a и P_b выглядит следующим образом

$$\{t(ch(P_a P_b),!a_1) \supset \Diamond[(t(ch(P_a P_b),?a_1)) \supset \Diamond(t(ch(P_b P_a),!a_2)) \supset \Diamond(t(ch(P_b P_a),?a_2))]\}$$

Здесь $t(Ch, A)$ – предикат, истинный, когда действие A поступает в канал Ch.

4. Проверка правильности взаимодействия агентов логическими программами

Как отмечалось в работе [1] перспективным путем автоматизации проверки правильности взаимодействия агентов является использование логического программирования, например, на основе языка логического программирования VISUAL PROLOG [7]. Иллюстрация структуры логической программы для проверки свойства обязательной реакции даже для сравнительно несложного примера агентов UAC и UAS в рамках настоящей стать слишком громоздка. Поэтому продемонстрируем проверку свойства обязательной реакции на простейшем примере, приведенном в предыдущем разделе. Предварительно введем понятия состояния процессов.

В примере предыдущего раздела поведение агентов P_a и P_b задано соответственно процессными выражениями $P_a = ch(P_a P_b)\,!a_1\, P_{a1},\ P_{a1} = ch(P_b P_a)\,?a_2.0$ и $P_b = ch(P_a P_b)\,?a_1\, P_{b1},\ P_{b1} = ch(P_b P_a)\,!a_2.0$. Перед началом взаимодействия процесс P_a находится в состоянии p_a, а после выдачи по собственной инициативе в канал $ch(P_a P_b)$ реакции $!a_1$ процесс P_a переходит в состояние p_{a1}. После получения по каналу $ch(P_b P_a)$ восприятия $?a_2$ в ответ на свою реакцию $!a_1$ процесс P_a прекращает свою работу (переходит к выполнению процесса 0 в состоянии p_{a0})). Аналогично процесс P_b до начала взаимодействия находится в состоянии p_b, ожидая в канале $ch(P_a P_b)$ восприятия $?a_1$. После этого восприятия процесс P_b переходит в состояние p_{b1} ожидания выполнения реакции $!a_2$. После выдачи в канал $ch(P_b P_a)$ реакции $!a_2$ процесс P_b прекращает свою работу (переходит к выполнению процесса 0 в состоянии p_{b0}).

Логическая программа проверки свойства обязательной реакции для нашего примера может выглядеть следующим образом.

```
domains
f =f(symbol, symbol, symbol)
 list = f*
predicates
 nondeterm transition(symbol, f, symbol)
 nondeterm accessible(symbol, list, symbol)
clauses
transition(pa, f(a,b,ia1), pa1).
transition(pa1,f(b,a,oa2), pa0).
transition(pb, f(a,b,oa1), pb1).
transition(pb1,f(b,a,ia2), pb0).

accessible(P1, [f(X,Y,Z)], P2) :- transition(P1, f(X,Y,Z), P2).
accessible(P1, [f(X,Y,Z)|Rest], P2) :- transition(P1, f(X,Y,Z), P3), accessible(P3, Rest, P2).
goal
accessible(pa,[f(a,b,ia1), f(b,a,oa2)], pa0),
accessible(pb,[f(a,b,oa1), f(b,a,ia2)], pb0).
```

Раздел *goal* содержит задачу нахождения двух последовательностей (списков) функторов, ведущих из начального состояния *pa* в состояние *pa0* и из начального состояния *pb* в состояние *pb0*, представляющих каналы и правильные последовательности реакций и восприятий, поступающие в них. Результатом работы данной программы является ответ “yes”, означающий выполнение свойства обязательной реакции для этого простого примера. В случае более сложных процессов эти последовательности могут содержать промежуточные функторы, представляющие другие каналы и действия.

5. Заключение

В настоящей статье рассмотрены принципы проверки правильности сетевого взаимодействия, представленного моделями последовательностных взаимодействующих процессов. В отличие от других работ здесь для описания процессов предлагается использовать канонический вариант языка, облегчающий формулировку требований правильности на языке временной модальной логики и переход к логической программе на языке логического программирования ПРОЛОГ. Приведен пример простой логической программы и результаты ее работы.

ЛИТЕРАТУРА

1. Девятков В.В., Мьё Т.Н. Формальный логический анализ корректности спецификаций сетевых SIP-протоколов. Инженерный журнал: наука и инновации, 2013, вып. 11.
2. Zave P. Understanding SIP Through Model-Checking. Proc. of the 2nd International Conference of Principles, Systems and Applications of IP Telecommunications. Springer-Verlag, 2008, vol. 5310, pp. 256–279.
3. Robin M. Communicating and Mobile Systems: The π-calculus. Cambridge. - UK: Cambridge University Press, 2003. 159 c.
4. Девятков В.В., Алфимцев А.Н. Необходимые и достаточные формальные свойства мультимодального интерфейса // Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение. Спец. вып. «Информационные технологии и компьютерные системы». М., 2011. С. 159-167.
5. Chellas B.F. Modal Logic an Introduction. The Press Syndicate of the University of Cambridge, 1980, 295 p.
6. Gabbay D., Hodkinson I., Reynolds M. Temporal Logic: mathematical founditions and computational aspects, vol. 1. Clarendon Press, Oxford, 1994.
7. Адаменко А.Н., Кучков А. Логическое программирование и Visual Prolog. Изд. БХВ – Петербург, 2003.
8. Девятков В.В., Алфимцев А.Н. Способ управления телевизором с помощью мультимодального интерфейса // Заявка на патент №2010103629, ФГУ ФИПС, Москва. 2010. 16 с.
9. Bishop S., Fairbairn M., Norrish M., Sewell P., Smith M., Wansbrough K. Rigorous specification and conformance testing techniques for network protocols, as applied to TCP, UDP and sockets. Proc. SIGCOMM′05. ACM, 2005, August.
10. Holzmann G.J. The Spin Model Checker: Primer and Reference Manual. Addison-Wesley, 2004, 596 p.
11. Девятков В.В., Сидякин И.М. Мультипроцессная система анализа телеметрической информации. Вестник МГТУ им. Н.Э. Баумана, Сер. Приборостроение, 2005, №4 (61), с. 56–85.
12. Девятков В.В. Построение, оптимизация и модификация процессов. Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение, 2012, №4, с. 60–79.

Myo Thet Naung
Bauman Moscow state technical university, Russia, Moscow
E-mail: komyothetnaung@gmail.com

Devyatkov Vladimir Valentinovich
Bauman Moscow state technical university, Russia, Moscow
E-mail: deviatkov@bmstu.ru

Formal validation networking agents, specifiable in the language of process expressions

Abstract. This article is dedicated to the development of methods of formal validation agent interaction modeling networking. The behavior of agents specified in the language of process expressions. Interacting agents are widely used to simulate network communications using different protocols. Automation validation specifications are encouraged to implement logic programs are prepared by the presented in this article methodology of process of transition from the canonical expressions that specify the behavior of agents, as well as the requirements of the correctness of the interaction in the language of modal logic. As a logical programming language to validate networking using Visual Prolog. This article develops the methodology outlined in the previous article the author. The main difference in the above article from the material already published the results is to use the graph does not describe the behavior of agents and canonical symbolic limiting expression to describe, but it facilitates the transition from a logic program and the correctness of the wording of claims in the language of modal logic. Methods of formal verification of correctness of networking illustrated by a simple example of the logic program on one of the earlier versions of Visual Prolog language, allows you to check the mandatory requirements of the reaction to the case of two interacting agents.

Keywords: agent; networking; modal logic; validation; process canonical expression; logic programming language