ВЕСТНИК ТОМСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

2023 Управление, вычислительная техника и информатика № 62

Tomsk State University Journal of Control and Computer Science

ИНФОРМАТИКА И ПРОГРАММИРОВАНИЕ

INFORMATICS AND PROGRAMMING

Научная статья
УДК 519.713
doi: 10.17223/19988605/62/12

Оптимизация клиентской компоненты по критерию отсутствия лишних диалогов с сервером

Максим Леонидович Громов[1], Светлана Анатольевна Прокопенко[2], Александр Павлович Сотников[3], Наталия Владимировна Шабалдина[4]

[1, 2, 3, 4] Томский государственный университет, Томск, Россия
[1] maxim.leo.gromov@gmail.com
[2] s.prokopenko@sibmail.com
[3] sotnikhtc@gmail.com
[4] nataliamailbox@mail.ru

Аннотация. Рассматривается задача оптимизации клиентской компоненты взаимодействующих по клиент-серверной архитектуре приложений. Критерием оптимизации выступает уменьшение количества взаимодействий, т.е. уточнение клиентской спецификации таким образом, чтобы по возможности не осуществлялось лишних запросов от клиента к серверу. Предлагается алгоритм модификации спецификации клиента по дереву достижимости.

Ключевые слова: оптимизация; клиент; сервер; конечный автомат; композиция.

***Для цитирования*:** Громов М.Л., Прокопенко С.А., Сотников А.П., Шабалдина Н.В. Оптимизация клиентской компоненты по критерию отсутствия лишних диалогов с сервером // Вестник Томского государственного университета. Управление, вычислительная техника и информатика. 2023. № 62. С. 107–114. doi: 10.17223/19988605/62/12

Original article
doi: 10.17223/19988605/62/12

Optimization of the client component by the criterion of the absence of redundant dialogues with the server

Maxim L. Gromov[1], Svetlana A. Prokopenko[2], Aleksandr P. Sotnikov[3], Natalia V. Shabaldina[4]

[1, 2, 3, 4] Tomsk State University, Tomsk, Russian Federation
[1] maxim.leo.gromov@gmail.com
[2] s.prokopenko@sibmail.com
[3] sotnikhtc@gmail.com
[4] nataliamailbox@mail.ru

Abstract. In this paper the problem of client component optimization within client-server interaction is considered. The optimization criterion is the reduction of client-server interactions number. Client's specification is clarified in way to minimize extra requests by client to server. The algorithm for modifying the client's specification according to the reachability tree is proposed.

Keywords: optimization; client; server; finite state machine; composition.

***For citation*:** Gromov, M.L., Prokopenko, S.A., Sotnikov, A.P., Shabaldina, N.V. (2023) Optimization of the client component by the criterion of the absence of redundant dialogues with the server. *Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie, vychislitelnaja tehnika i informatika – Tomsk State University Journal of Control and Computer Science*. 62. pp. 107–114. doi: 10.17223/19988605/62/12

Введение

Клиент–сервер – классическая архитектура организации информационных систем. В этой архитектуре система представляется двумя программами: клиентом и сервером. Сервер владеет ресурсами (файлами, базами данных и т.п.) и обслуживает запросы на доступ к ним. Клиент запускается, например, в браузере пользователя информационной системы и через свой графический интерфейс (кнопки, текстовые поля и т.п.) общается с пользователем. Между собой клиент и сервер общаются через телекоммуникационную сеть (например, Интернет). Всякое сообщение (запрос) от клиента серверу или наоборот проходит через сеть. В общем случае путь сообщения пролегает через некоторое количество промежуточных узлов (маршрутизаторов, коммутаторов и т.д.), и каждый узел затрачивает энергию на обработку сообщения. Кроме того, время, затрачиваемое на прохождение запроса к серверу, формирование ответа и прохождение ответа от сервера к клиенту, имеет значительные величины, заметные человеку. Поэтому манипуляции с интерфейсом в браузере, например нажатия кнопки, которые требуют взаимодействия клиента с сервером, выглядят с точки зрения человека как некоторое «подтормаживание» системы, что ухудшает впечатление пользователя (user experience) от системы. Таким образом, уменьшение количества взаимодействий между клиентом и сервером является задачей, актуальной как с точки зрения экологии (уменьшении совокупного потребления ресурсов системой), так и с точки зрения пользовательского опыта.

При совместной работе клиентской и серверной реализаций протоколов возникают ситуации, когда клиент осуществляет лишние (избыточные) запросы к серверу, в то время как и без этих запросов ответ клиента конечному пользователю предопределен. Такая предопределенность может быть обусловлена как требованиями RFC, так и, в некоторых случаях, накопленной к этому этапу взаимодействия информацией, например о настройках сервера или о реализованных на сервере опциях.

Задача модификации («ограничения») спецификации клиентского приложения, правда, с несколько иной целью (по иному критерию), рассматривается в работе [1]. Предлагается ограничить спецификацию клиента, чтобы на серверное приложение не могли поступить входные последовательности, на которых поведение сервера не определено. Отметим, что в этой работе дана только идея; шаги определения недопустимой последовательности и ограничения спецификации клиента описаны общими формулировками.

Распространенным подходом к оптимизации одной из взаимодействующих компонент является решение автоматных уравнений [2, 3] и неравенств, после чего среди всего множества решений выбирается такое решение, которое соответствует заданному критерию. В нашем случае данный подход тоже можно было бы исследовать. Тогда необходимо решить уравнение

$$X \lozenge Server \cong Spec,$$

где *Server* – спецификация сервера, *Spec* – спецификация взаимодействия клиента и сервера, которая может быть получена в результате композиции исходных спецификаций клиента и сервера, $\lozenge$ – операция композиции, $\cong$ – отношение эквивалентности. Среди всех решений данного уравнения нужно выбрать такое решение X (такого клиента), которое порождает минимум диалогов с сервером. Возможно, такая постановка задачи окажется тесно связанной с поиском так называемых l-ограниченных решений [2].

В данной работе предлагается оптимизировать спецификацию клиента с точки зрения отсутствия лишних запросов к серверу путем построения модифицированного дерева достижимости. Такое дерево достижимости отличается от классического, введенного в работе [4] и используемого с некоторыми модификациями, например в работе [5]. В предлагаемом в данной работе варианте дерева достижимости вершинам ставится в соответствие помимо пар состояний клиента и сервера еще и таблица пе-

реходов-выходов сервера, уточненная согласно уже пройденному пути. В нашей постановке задачи спецификация сервера является недетерминированным автоматом, и при построении дерева некоторые ребра соответствуют фиксации переходов в таблице переходов-выходов сервера, поскольку мы полагаем, что реализация является детерминированной. Также в предлагаемом в работе модифицированном дереве достижимости стабильной объявляется не всякая вершина, при переходе в которую был выдан внешний выходной символ, а только такая, которая уже встречалась выше с той же таблицей переходов-выходов сервера.

1. Основные понятия и определения

1.1. Конечный автомат

Конечным автоматом называется пятерка $S = (S, I, O, T_S, s_0)$, где S – непустое конечное множество состояний с выделенным начальным состоянием s_0, I – непустое множество входных символов, называемое входным алфавитом, O – непустое множество выходных символов, называемое выходным алфавитом, $T_S \subseteq I \times S \times S \times O$ – отношение переходов [2].

Автомат S называется полностью определенным, если для каждой пары $(i, s) \in I \times S$ существует по крайней мере одна пара $(o, s') \in O \times S$ такая, что $(i, s, s', o) \in T_S$. В противном случае автомат S называется частичным.

Автомат S называется наблюдаемым, если для любой тройки $(i, s, o) \in I \times S \times O$ существует не более одного состояния $s' \in S$ такого, что $(i, s, s', o) \in T_S$, в противном случае автомат называется ненаблюдаемым (т.е. в наблюдаемом автомате, зная текущее состояние, входное воздействие и наблюдая выходную реакцию, можно однозначно определить следующее состояние). Поскольку известно, что для любого недетерминированного ненаблюдаемого автомата существует эквивалентная наблюдаемая форма [6], для удобства далее будем рассматривать только наблюдаемые автоматы.

Автомат S называется детерминированным, если для любой пары $(i, s) \in I \times S$ существует не более одной пары $(o, s') \in O \times S$ такой, что $(i, s, s', o) \in T_S$, в противном случае автомат называется недетерминированным.

В настоящей работе мы рассматриваем полностью определенные, возможно, недетерминированные, но всегда наблюдаемые автоматы.

Отношение переходов обычным образом распространяется на входные и выходные последовательности. Пара α/β, где $\alpha = i_1 \ldots i_k$, $\beta = o_1 \ldots o_k$, называется вход-выходной последовательностью автомата в состоянии s, если существует последовательность состояний $s_0, ..., s_k \in S$ такая, что $s_0 = s$ и $(i_j, s_{j-1}, s_j, o_j) \in T_S$, $j = 1, ..., k$. Множество вход-выходных последовательностей в состоянии s называется *языком автомата в состоянии* s, обозначение: $L_S(s)$. Язык автомата в начальном состоянии называется просто *языком* автомата и обозначается L_S. Автомат $A = (S, I, O, T_A, s_0)$ есть *редукция* автомата $B = (Q, I, O, T_B, s_0)$ (обозначение: $A \le B$), если $L_A \subseteq L_B$. Автоматы A и B *эквивалентны* (обозначение: $A \cong B$), если $A \le B$ и $B \le A$.

1.2. Параллельная композиция конечных автоматов

Рассмотрим параллельную композицию автоматов *client* $= (S, X \cup V, Y \cup U, T_{client}, s_0)$ и *server* $= (Q, U, V, T_{server}, q_0)$ (рис. 1), в которой:

– автомат *client* имеет входной алфавит $X \cup V$, выходной алфавит $Y \cup U$;

– автомат *server* имеет входной алфавит U, выходной алфавит V;

– алфавиты X_1, X_2, Y_1, Y_2, V и U попарно не пересекаются.

Параллельная композиция автоматов *client* и *server* имеет:

– входной алфавит X;

– выходной алфавит Y;

– под действием входного символа $x \in X$ композиция «вырабатывает» внешний выходной символ $y \in Y$ или внутренний выходной символ из алфавита U, который является входным для другой компоненты;

– следующий входной символ может быть подан на композицию только после того, как компоненты закончили внутренний диалог.

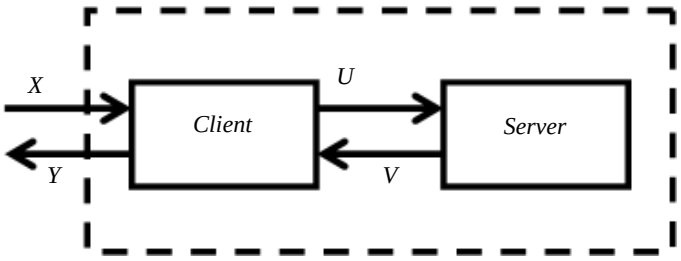

Рис. 1. Схема взаимодействия клиента и сервера
Fig. 1. Scheme of interaction between client and server

Иными словами, параллельная композиция представляет собой пару компонент, взаимодействующих в режиме диалога, где лишь внешняя компонента (*client*) может получать входные воздействия от среды (пользователя или тестировщика), в то время как внутренняя компонента (*server*) принимает внутренние выходные символы от первой компоненты и возвращает выходные символы, которые являются входными символами для внешней компоненты. Для того чтобы анализировать совместную работу таких компонент, их поведение часто моделируется одним конечным автоматом-композицией.

Диалогом считается непустое взаимодействие компонент, которое происходит между подачей внешнего входного символа и выдачей внешней выходной реакции.

Отметим, что при совместной работе компоненты могут попадать в бесконечный диалог, при котором никогда не достигается новое стабильное состояние. Такая ситуация называется осцилляцией, или зацикливанием. Также при совместной работе компонент на компоненту может быть подано входное воздействие, при котором поведение компоненты не определено. Такие ситуации в данной работе не рассматриваются.

2. Построение модифицированного дерева достижимости

Ведем понятие модифицированного дерева достижимости, отличающегося от классического дерева достижимости (reachability graph) следующим образом.

В классическом дереве достижимости [4] вершинам сопоставляются так называемые глобальные состояния – пары состояний взаимодействующих автоматов. Ребра – переходы между этими состояниями, помечены парой «входное воздействие / выходная реакция» в алфавитах активной на данный момент компоненты. В параллельной композиции компоненты активны по очереди, и ребро будет помечено парой x/y, если на внешнюю компоненту (на *client*) был подан внешний входной символ $x \in X$ и клиент ответил внешним выходным символом $y \in Y$ (диалога между клиентом и сервером в этом случае не произошло). Если же в ответ на внешний входной символ $x \in X$ клиент начинает диалог с сервером, то ребро будет помечено парой x/u, где $u \in U$. Символ $u \in U$ является одновременно выходным символом для клиента и входным символом для сервера, значит, следующее ребро будет помечено парой u/v – активной станет компонента *server*. Сервер является внутренней компонентой в данной композиции, поэтому ответ во внешнюю среду выдан быть не может. Ответ приходит на клиентскую компоненту. Дальше снова активным становится клиент, поскольку $v \in V$ – входное воздействие на компоненту *client*. Теперь клиентская компонента может либо снова ответить выходным символом из алфавита U, и тогда ребро будет помечено парой v/u, что продолжит диалог между клиентом и сервером, либо клиент ответит внешним выходным символом $y \in Y$, ребро будет помечено парой v/y, автомат композиции перейдет в так называемой *стабильное состояние*. Напомним, что следующий

входной символ может быть подан на композицию только после того, как компоненты закончили внутренний диалог – это и есть стабильное состояние. Таким образом, построение дерева достижимости осуществляется путем моделирования взаимодействия автоматов-компонент. Корнем дерева является пара начальных состояний автоматов-компонент. Моделирование ведется по всем входным символам из внешнего алфавита X. Вершина объявляется листом, если она соответствует стабильному состоянию и уже встречалась. Построение дерева достижимости заканчивается, когда «раскрыты» все нелистовые вершины, которым соответствуют стабильные состояния.

В модифицированном дереве каждая вершина дополняется таблицей переходов-выходов сервера (внутренней компоненты). Напомним, что серверная компонента в нашем случае является недетерминированным автоматом. Пусть рассматривается нестабильное глобальное состояние st, и активной компонентой является сервер. Пусть в предыдущем состоянии на вход сервера был подан символ u. Тогда, раскрывая состояние st, мы перебираем все пары u/v, такие что из состояния t сервера есть переход по входу u с выходом v в таблице переходов-выходов, приписанной рассматриваемой вершине. Из текущей вершины (которой соответствует st) в дерево добавляем ребро с пометкой u/v в новую вершину st' (в таблице переходов-выходов текущей вершины есть переход из t в t' по u с выдачей v – в новую вершину в качестве пометки добавляем таблицу переходов-выходов из текущей вершины, но удалив в ней все остальные переходы из t по u, кроме выбранного нами для данного ребра и новой вершины перехода). Вершина объявляется листом, если ее пометка соответствует стабильному состоянию и уже встречалась в дереве именно как стабильное состояние с такой пометкой (обратим внимание: таблицы переходов-выходов у этих вершин должны совпадать). Построение дерева достижимости также заканчивается, когда «раскрыты» все нелистовые вершины, которым соответствуют стабильные состояния.

Если достигнуто стабильное состояние и не изменилась таблица переходов-выходов сервера (соответствующая данной паре состояний), то это стабильное состояние объявляется листом. Заканчиваем построение дерева достижимости, если все нераскрытые стабильные состояния объявлены листьями.

В качестве примера рассмотрим автоматы клиента и сервера, представленные на рис. 2. Модифицированное дерево достижимости представлено на рис. 3.

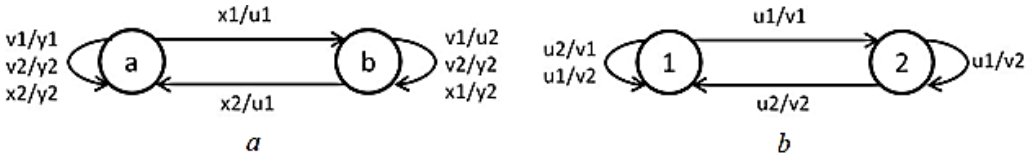

Рис. 2. Автомат клиента (*a*) и автомат сервера (*b*)
Fig. 2. Client's Finite State Machine (*a*) and Server's Finite State Machine (*b*)

Можно видеть, что, как было описано выше, вершинам сопоставлены не только пары состояний автоматов *client* и *server*, но и обозначения $T0$, $T1$, $T2$, соответствующие трем различным таблицам переходов-выходов сервера (таблица).

Вершину дерева, соответствующую стабильному состоянию автомата композиции, назовем *стабильной вершиной*. Путь в дереве между двумя стабильными вершинами длины более одного, не проходящий через другие стабильные вершины, назовем *диалоговым путем*. Диалоговый путь, такой что в нем состояние сервера и таблица переходов-выходов сервера в начале пути совпадают с состоянием сервера и таблицей переходов-выходов сервера в конце пути, назовем *избыточным*.

Напомним, что идея модификации клиента заключается в том, чтобы избежать лишних диалогов между клиентом и сервером; такие диалоги на дереве соответствуют избыточным диалоговым путям. Тут нужно заметить, что значительная часть диалогов должна быть сохранена, т.е. в попытке оптимизировать мы не должны потерять «полезное» взаимодействие. В частности, нельзя избавляться от диалога, если в процессе диалога сервер изменил свое состояние.

Дуги в виде пунктирных линий, иллюстрируют работу алгоритма 1, приведенного ниже.

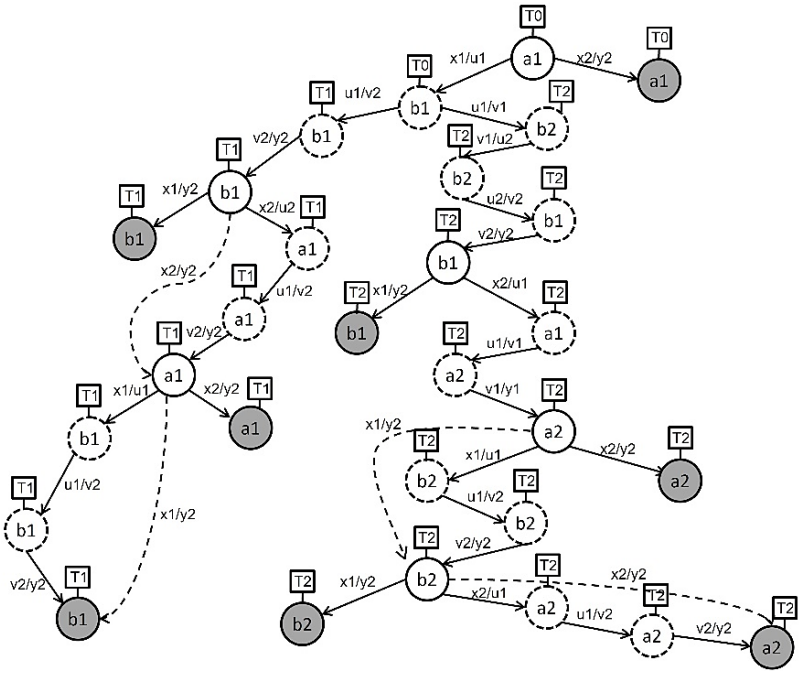

Рис. 3. Модифицированное дерево достижимости, построенное по автоматам *client* и *server*
Fig. 3. Modified reachability tree based on *client* and *server* automata

Переходы-выходы сервера

*T*0	1	2
*u*1	1/*v*2 2/*v*1	2/*v*2
*u*2	1/*v*1	1/*v*2

*T*1	1	2
*u*1	2/*v*1	2/*v*2
*u*2	1/*v*1	1/*v*2

*T*2	1	2
*u*1	1/*v*2	2/*v*2
*u*2	1/*v*1	1/*v*2

Алгоритм 1 построения модифицированной спецификации клиента по дереву достижимости.

Вход: Модифицированное дерево достижимости.

Выход: Автомат клиента, оптимизированный с точки зрения отсутствия лишних взаимодействий с сервером.

Шаг 1. Все избыточные пути в дереве «схлопываем» (оставляем входной символ первого ребра пути и выходной символ последнего ребра пути).

Шаг 2. Ставим в соответствие всем стабильным вершинам с одинаковыми пометками одно состояние клиента. Получим подмножество Z множества состояний нового клиента. Это подмножество будет дополнено промежуточными вершинами в процессе формирования графа переходов-выходов клиента.

Шаг 3. Начинаем перебирать все состояния из множества Z. Двигаемся по дереву.

Для каждого состояния $z \in Z$ определяем поведение под действием каждого внешнего входного символа x:

a) если вершина, соответствующая состоянию z, является началом ребра, помеченного парой x/y, $y \in Y$ и приводящего в вершину, соответствующую состоянию z', то добавляем в автомат переход (x, z, z', y);

b) если вершина, соответствующая состоянию z, является началом ребра, помеченного парой x/u, $u \in U$, то определяем переход из состояния z в новое состояние $s \notin Z$, помеченный парой x/u. Далее в состоянии s определяем переход под действием входного символа $v \in V$ следующим образом:

– если вершина, соответствующая состоянию s, является началом ребра, помеченного парой v/y, $y \in Y$, то в автомат добавляется переход из состояния s в одно из состояний множества Z (согласно переходу в дереве), помеченный парой v/y;

– в противном случае повторяем шаг 3b.

В результате применения алгоритма автомат может получиться неприведенным, частичным, недетерминированным; число состояний может быть больше, чем у исходного автомата клиента.

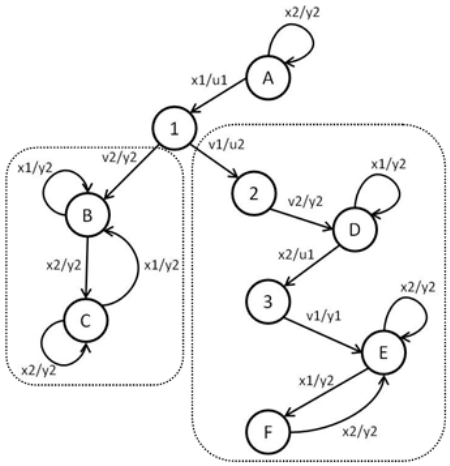

Рис. 4. Автомат клиента, построенный по алгоритму 1
Fig. 4. Client's Finite State Machine derived according to the Algorithm 1

На рис. 4 представлен автомат клиента, построенный по алгоритму 1. Пунктиром выделены различные варианты поведения клиента в зависимости от того, какую реакцию клиент получает от сервера в предшествующем диалоге. Отметим, что полученный в результате применения алгоритма 1 автомат клиента является частичным и неприведенным.

Заключение

В работе исследуется задача оптимизации спецификации клиентской компоненты взаимодействующих по клиент-серверной архитектуре приложений. Предложено в качестве критерия использовать отсутствие избыточных запросов к серверу. Оптимизация клиентской компоненты осуществляется путем построения модифицированного дерева достижимости и построения на основе этого дерева новой спецификации клиента. Отметим, что при решении задачи оптимизации число состояний клиента может увеличиться, кроме того, спецификация клиента может стать недетерминированным, частичным, неприведенным автоматом.

Результаты данной работы могут быть использованы при проектировании архитектуры информационных систем.

Список источников

1. Shirokova E. Checking Robustness of Web Services Based on the Parallel Composition of Partial Timed Finite State Machine // 2018 IEEE East-West Design & Test Symposium (EWDTS). 2018. P. 1–6.
2. Евтушенко Н.В., Рекун М.В., Тихомирова С.В. Недетерминированные автоматы: анализ и синтез : учеб. пособие. Томск : Том. гос. ун-т, 2009. Ч. 2: Решение автоматных уравнений. 111 с.
3. Castagnetti G., Piccolo M., Villa T., Yevtushenko N., Mishchenko A., Brayton R.K. Solving Parallel Equations with BALM-II : Technical Report No. UCB/EECS2012-181 // Electrical Engineering and Computer Sciences University of California at Berkeley. 2012. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-181.pdf (Дата обращения: 20.08.2022).
4. West C.H. An automated technique of communication protocols validation // IEEE Trans. Comm. 1978. V. 26. P. 1271–1275.
5. El-Fakih K., Trenkaev V., Spitsyna N., Yevtushenko N. FSM Based Interoperability Testing Methods for Multi Stimuli Model // Testing of Communicating Systems : 16th International Conference, TestCom 2004 : Proc. Oxford, 2004. P. 60–75. (Lecture Notes in Computer Science; v. 2978).
6. Starke P.H. Abstract Automata. New York : American Elsevier Publishing Company, 1972. 419 p.

References

1. Shirokova, E. (2018) Checking Robustness of Web Services Based on the Parallel Composition of Partial Timed Finite State Machines. *2018 IEEE East-West Design & Test Symposium (EWDTS)*. pp. 1–6. DOI: 10.1109/EWDTS.2018.8524850

2. Evtushenko, N.V., Rekun, M.V. & Tikhomirova, S.V. (2009) *Nedeterminirovannye avtomaty: analiz i sintez* [Nondeterministic finite state machines]. Vol. 2. Tomsk: Tomsk State University.
3. Castagnetti, G., Piccolo M., Villa, T., Yevtushenko, N., Mishchenko, A. & Brayton, R.K. (2012) Solving Parallel Equations with BALM-II. *Technical Report No. UCB/EECS2012-181*. Berkeley: Electrical Engineering and Computer Sciences University of California. [Online] Available from: http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-181.pdf. (Accessed: 20th August 2022).
4. West, C.H. (1978) An automated technique of communication protocols validation. *IEEE Trans. Comm.* 26. pp. 1271–1275.
5. El-Fakih, K., Trenkaev, V., Spitsyna, N. & Yevtushenko, N. (2004) FSM Based Interoperability Testing Methods. *Proceedings of the IFIP 16th International Conference on Testing of Communicating Systems.* 2978. pp. 60–75.
6. Starke, P.H. (1972) *Abstract Automata*. New York: American Elsevier Publishing Company.

***Информация об авторах*:**

Громов Максим Леонидович – доцент, кандидат физико-математических наук, доцент кафедры информационных технологий в исследовании дискретных структур Национального исследовательского Томского государственного университета (Томск, Россия). E-mail: maxim.leo.gromov@gmail.com

Прокопенко Светлана Анатольевна – доцент, кандидат технических наук, доцент кафедры информационных технологий в исследовании дискретных структур Национального исследовательского Томского государственного университета (Томск, Россия). E-mail: s.prokopenko@sibmail.com

Сотников Александр Павлович – аспирант кафедры информационных технологий в исследовании дискретных структур Национального исследовательского Томского государственного университета (Томск, Россия). E-mail: sotnikhtc@gmail.com

Шабалдина Наталия Владимировна – доцент, кандидат технических наук, доцент кафедры информационных технологий в исследовании дискретных структур Национального исследовательского Томского государственного университета (Томск, Россия). E-mail: nataliamailbox@mail.ru

Вклад авторов: все авторы сделали эквивалентный вклад в подготовку публикации. Авторы заявляют об отсутствии конфликта интересов.

Information about the authors:

Gromov Maxim Leonidovich (Candidate of Physical and Mathematical Sciences, Associate Professor, National Research Tomsk State University, Tomsk, Russian Federation). E-mail: maxim.leo.gromov@gmail.com

Prokopenko Svetlana Anatolievna (Candidate of Technical Sciences, Associate Professor, National Research Tomsk State University, Tomsk, Russian Federation). E-mail: s.prokopenko@sibmail.com

Sotnikov Aleksandr Pavlovich (Post-graduate Student, National Research Tomsk State University, Tomsk, Russian Federation). E-mail: sotnikhtc@gmail.com

Shabaldina Natalia Vladimirovna (Candidate of Technical Sciences, Associate Professor, National Research Tomsk State University, Tomsk, Russian Federation). E-mail: nataliamailbox@mail.ru

Contribution of the authors: the authors contributed equally to this article. The authors declare no conflicts of interests.

Received 26.09.2022; accepted for publication 01.03.2023

Поступила в редакцию 26.09.2022; принята к публикации 01.03.2023