УДК 004.4
DOI 10.21685/2072-3059-2019-3-5

А. С. Бождай, Ю. И. Евсеева, Д. В. Артамонов

ИСПОЛЬЗОВАНИЕ МАШИННОГО ОБУЧЕНИЯ С ПОДКРЕПЛЕНИЕМ В СОЗДАНИИ САМОАДАПТИВНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ[1]

Аннотация.

Актуальность и цели. Создание эффективных методов самоадаптации программных систем является достаточно обсуждаемой темой в сообществе разработчиков программного обеспечения. Актуальность проблемы обусловлена высокой сложностью современных программных систем, существенными затратами на их разработку и сопровождение. Указанная проблема сложности непосредственно сказывается на качестве функционирования программных комплексов, поскольку заранее достаточно сложно учесть весь спектр ситуаций, в которых поведенческий профиль программы окажется неадекватным или неточным. Схожая проблема связана и с оптимальностью функционирования подобных систем: программное обеспечение должно самостоятельно определять ситуации, в которых требуется увеличить или снизить количество потребляемых ресурсов для оптимизации производительности. Целью работы является поиск универсальной техники самоадаптации программных систем, способной решить проблему повышения качества функционирования программного обеспечения.

Материалы и методы. В качестве основного метода решения поставленной задачи применен модифицированный метод машинного обучения с подкреплением. Отличие от классического метода заключается в том, что лежащая в его основе процедура является модельно-ориентированной (в ее основе лежит модель предметной области, в которой функционирует система) и учитывает фактор возникающих в процессе эксплуатации программного обеспечения неопределенностей.

Результаты. К основным результатам работы следует отнести: обзор и классификацию существующих методов реализации самоадаптивного поведения программных систем; обоснование возможности применения модифицированной процедуры машинного обучения с подкреплением при разработке самоадаптивного программного обеспечения, функционирующего на основе модели предметной области и учитывающего фактор динамики возникающих неопределенностей.

Выводы. Рассмотренная процедура обучения с подкреплением применима к широкому кругу адаптивных программных систем. Ее применение позволит решить проблемы, связанные с поведением сложных программных систем: оптимальность потребления ресурсов в различных ситуациях, неопределенности, возникающие в процессе эксплуатации программы.

Ключевые слова: машинное обучение, обучение с подкреплением, самоадаптируемое программное обеспечение.

[1] Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 18-07-00408.

A. S. Bozhday, Yu. I. Evseeva, D. V. Artamonov

USING REINFORCEMENT LEARNING IN CREATING SELF-ADAPTIVE SOFTWARE

Abstract.

Background. Creating effective software self-adaptation techniques is a fairly discussed topic in the software development community. The urgency of the problem is due to the high complexity of modern software systems, the significant costs of their development and maintenance. The problem of the high complexity of the system also raises the problem of the quality of its functioning: in the case of software systems having a complex structure and behavior, it is rather difficult to take into account the whole spectrum of situations in which they will exhibit undesirable behavior for users. A similar problem is associated with the optimal functioning of such systems: the software must independently determine situations in which it is necessary to increase or decrease the amount of resources consumed in order to optimize performance. Thus, the aim of the work is to create a new universal technique for self-adaptation of software systems that can solve the problem of improving the quality of software functioning.

Materials and methods. As the main method for solving the problem, a modified method of training with reinforcement was used. The difference from the classical method is that the procedure underlying it is model-oriented (it is based on the domain model in which the system operates) and takes into account the factor of uncertainties arising during the operation of the software.

Results. The main results of the work include: a review and classification of existing methods for implementing self-adaptive behavior of software systems; a modified learning procedure with reinforcement, which operates on the basis of the model of the subject area of the system and takes into account the factor of arising uncertainties.

Conclusions. The proposed reinforcement learning procedure is applicable to a wide range of software systems. Its application will solve the problems associated with the behavior of complex software systems: the optimality of resource consumption in various situations, the uncertainties that arise during the operation of the program.

Keywords: machine learning, reinforcement learning, self-adaptive software.

Введение

Проблема повышения эффективности и надежности функционирования программного обеспечения (ПО), а также сокращения затрат на его разработку и сопровождение в настоящее время является весьма актуальной. Актуальность обусловлена высокой сложностью современных программных систем и большими ресурсозатратами на их разработку и сопровождение. Проблема сложности ПО порождает проблему снижения качества его функционирования. Очевидно, что заранее сложно учесть, как поведет себя система при всех возможных внешних условиях. Программа, показывающая хорошие (например в плане производительности) результаты в одних ситуациях, может недостаточно хорошо работать в других. Кроме того, разработка и поддержка сложной программной системы, предусматривающей функционирование в различном программно-аппаратном окружении, при различных внешних условиях (в частности пользовательских) является длительной и ресурсозатратной задачей.

Решением перечисленных проблем является поиск новых методов самоадаптации программных систем. Большим потенциалом в данном случае обладает широко известная и применяемая во многих сферах деятельности технология машинного обучения. Особенностью методов машинного обучения является то, что они позволяют использовать уже имеющийся опыт решения определенных задач для планирования дальнейшего поведения системы. Любое ПО, построенное на методах машинного обучения, будет способно самостоятельно анализировать собственное поведение и делать выводы о том, какие действия и при каких условиях оно должно выполнять.

1. Обзор существующих методов реализации самоадаптивного поведения в программных системах

Для реализации собственного адаптивного поведения программная система должна брать информацию из окружающей среды и анализировать ее. Реализовать такой процесс достаточно трудно, так как спецификации обычно являются неполными, окружающая среда постоянно меняется, даже тщательно разработанные проекты порой опираются на предположения, которые могут терять актуальность. Существует несколько групп инструментов, которые позволяют преодолевать данные сложности.

Первой группой инструментов являются языки динамического программирования. В отличие от статических языков, таких как Си, языки вроде Dylan и CLOS позволяют разрабатывать программные интерфейсы, способные меняться в процессе выполнения. Например, в Dylan работающая программа может добавить метод к существующему классу без доступа к исходному коду. Другим популярным языком, включающим в себя динамические элементы, является Java. Созданию специфического динамического языка программирования самоадаптивных приложений посвящена, например, работа [1].

Другим подходом к разработке адаптивных программных систем является использование агентных технологий. Программный агент – это автономный процесс, способный реагировать на среду исполнения и вызывать изменения в среде исполнения, возможно, в кооперации с пользователями или другими агентами. Программные агенты обладают некоторыми знаниями об окружающем мире, которые позволяют им самим решать проблемы без вмешательства пользователя. Однако агент никогда не имеет полного контроля над окружающей средой. Хотя агентные технологии довольно популярны в настоящее время и активно используются для создания адаптивного программного обеспечения [2], существует несколько сложностей в их использовании. Первая заключается в том, что пользователь не всегда может получить то, что хочет, по той причине, что в рамках сложной среды не сразу удается полностью автоматизировать все процессы. Весьма трудно получить программу, которая будет одинаково хорошо удовлетворять критериям качества, быстродействия и ресурсоемкости. Поэтому необходимо предоставить пользователю возможность самостоятельно решить, какие критерии функционирования ПО для него важны в тех или иных ситуациях. Второй проблемой является возможная нечеткость в получаемых результатах. Традиционные программы построены на булевой логике, которая рассматривает исход любого логико-вычислительного процесса как истину или ложь. В реальном же

мире все события происходят с некоторой долей вероятности, и адекватные программные модели должны учитывать этот факт. Третья сложность заключается в том, что окружение могут изменить другие программы, с которыми также, возможно, придется конкурировать или обмениваться данными.

Третью группу методов составляют методы теории принятия решений. Теория принятия решений находится на стыке теории полезности и теории вероятностей, и ее можно использовать в качестве языка для описания адаптивных программных систем. Чтобы реагировать на пользовательские предпочтения и справляться с неопределенностью, теория предоставляет необходимый математический аппарат. Также методы теории могут быть полезны при разработке способов коммуникации между программными агентами. Применению методов теории принятия решений посвящена работа [3].

Четвертую группу методов составляют методы на основе обучения с подкреплением. Обучение с подкреплением представляет собой способ машинного обучения, в ходе которого испытуемая система обучается, взаимодействуя с некоторой средой. Откликом среды на принятые решения являются сигналы подкрепления, поэтому такое обучение является частным случаем обучения с учителем, но учителем является среда или ее модель. Применению обучения с подкреплением в разработке самоадаптивного программного обеспечения посвящена работа [4].

Последней группой методов являются методы, основанные на применении вероятностных сетей (также известных как байесовские сети, сети принятия решений, диаграммы влияния). Вероятностная сеть – это графовая вероятностная модель, представляющая собой множество переменных и их вероятностных зависимостей. Существуют алгоритмы для поиска значений переменных, изучения качественных и количественных зависимостей в данных, вычисления ценности информации. Одной из работ, посвященных применению вероятностных сетей в разработке самоадаптивных программных систем, является [5]. В данной работе авторами предлагается использовать расширенную технику обучения с подкреплением – модельно-ориентированный метод. Применение подобной модификации позволит программной системе в процессе самоадаптации учитывать специфику собственной предметной области, в частности, возможные возникающие в процессе функционирования неопределенности.

2. Обучение с подкреплением и самоадаптивное программное обеспечение

Как было показано в предыдущем разделе, при использовании машинного обучения с подкреплением испытуемая система, называемая также агентом, обучается, взаимодействуя со средой. Взаимодействие со средой обычно осуществляется на протяжении некоторого количества временных шагов. На каждом шаге агент наблюдает за состоянием среды и на основе этого наблюдения выбирает определенное действие. На следующем шаге агент получает вознаграждение (некоторое численное значение) за совершенное действие и переходит в новое состояние. Взаимодействие между агентом и средой в программных системах можно охарактеризовать как планирование (рис. 1). Самодаптивная программа в таком случае будет играть роль агента, наблюдая за текущим состоянием среды и используя для этого собственные датчики. При

этом вознаграждение обычно рассчитывается путем измерения различных значимых факторов и формирования функции полезности.

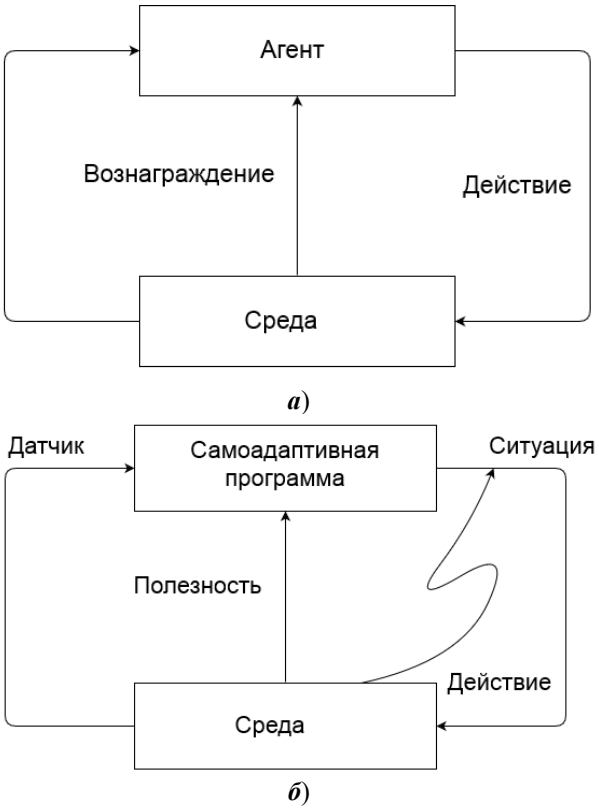

Рис. 1. Взаимодействие системы и среды в задаче обучения с подкреплением (***а***) и задаче самоадаптации программной системы (***б***)

Под «ситуацией» здесь понимается некоторое дискретное изменение комплекса параметров, приводящее к качественным изменениям в состояниях среды. Это один из центральных аспектов разработки самоадаптивного ПО, который остается неучтенным в классическом обучении с подкреплением [6]. Точное отслеживание момента наступления ситуации позволяет системе узнать, когда именно она должна оценивать собственное состояние и получать вознаграждение, а также предпринимать действие, соответствующее текущему состоянию. Ситуация предоставляет агенту информацию об окружающей среде таким образом, чтобы он вел себя адекватно текущим параметрам среды. Иными словами, ситуация накладывает ограничения на множество доступных к выбору действий. В качестве примера можно рассмотреть суперкомпьютер, способный управлять распределением собственных ресурсов. В ситуации увеличения рабочей нагрузки следует увеличить количество ресурсов для повышения производительности. И, напротив, количество ресурсов должно быть уменьшено для снижения энергопотребления при уменьшении рабочей нагрузки.

Для применения обучения с подкреплением к самоадаптивному ПО необходимо расширить определение классического марковского процесса принятия решений, дополнив его определением «ситуация».

Пусть $I = \{i_1, i_2, ..., i_n\}$ – конечное множество ситуаций. Ситуация $i_i \in I$ определяет состояние среды или внутреннее событие, с которым может столкнуться агент, и запускает его адаптацию.

Тогда $A|_{s_i, i_i}$ указывает на множество возможных действий, которые агент может предпринять в состоянии s_i и ситуации i_i. Пусть $S*$ будет комбинацией между множеством состояний S и множеством ситуаций I. Тогда марковский процесс принятия решений можно представить четверкой $\{S*, A, R, T\}$, где A – конечное множество действий; R – функция вознаграждения, получаемого за переходы между состояниями; T – функция, возвращающая вероятность того, что при определенном действии система изменит свое состояние. Подобное определение марковского процесса позволяет без каких-либо затруднений использовать обучение с подкреплением при создании самоадаптивных программных систем. Концепция ситуации дает возможность системным инженерам проектировать хранилище системных политик, дополняя его еще одним измерением в соответствии со стандартами самоадаптивных систем, а также способствует глобальной унификации систем.

3. Процесс построения модели

Модельно-ориентированное обучение с подкреплением обеспечивает стабильную производительность путем поддержки явной модели операционной среды. Следовательно, преобразование инженерных знаний в такую модель дает сильный выигрыш при обучении. Само построение модели является специфической проблемой для каждой предметной области, однако и в этом процессе можно выделить некоторые общие для любого домена черты. В данной работе мы обсудим технический процесс построения модели на основе анализа требований и спецификаций программной системы. Он будет представлять собой следующую 4-этапную процедуру:

1. Определение системных состояний, действий и ситуаций.
2. Анализ контекста и моделирование неопределенностей.
3. Отображение неопределенностей на действия.
4. Получение переходов между состояниями.

Определение состояний и действий является весьма важным этапом в создании самоадаптирующейся системы. На ранних стадиях разработки аналитики, работая с пользовательскими требованиями, определяют цели и сценарии работы системы. Исследований, посвященных данному этапу создания программных систем, в настоящее время насчитывается довольно много. Например, в работе [7] описана процедура представления и анализа сценариев работы программы для обнаружения состояний, действий и параметров системы. Состояние обычно представляет собой комбинацию важных условий функционирования системы, в то время как действие – это функция либо процесс, который приводит систему к переходу из одного состояния в другое. Наконец, ситуация – это событие, которое запускает сценарий адаптации агента. После этого шага можно получить первоначальную модель смены состояний.

На втором этапе производится анализ контекста выполнения системы с целью моделирования неопределенностей, которые могут как-либо повлиять на работу программы. В последнее десятилетие ряд исследователей при-

ложили немалые усилия в борьбе с неопределенностями в динамических средах. В ранних исследованиях, например в [8], использовалась в основном концепция исключений, поздние же работы были посвящены более подробному анализу проблемы и разработке оригинальных методов ее решения. Например, в работе [9] впервые представлена классификация неопределенностей и выделены следующие их основные категории: неточности в модели, шумы, неточность параметров, человеческий фактор, децентрализация. Данная классификация вполне пригодна для определения источников неопределенностей в рассматриваемой нами задаче. Кроме того, на этапах проектирования системы (в частности, на этапе функционального проектирования) во время анализа зависимостей между функциями и компонентами необходимо принимать во внимание и некоторые другие факторы структуры и реализации системы, которые также могут оказывать влияние на производимые ею действия. Этот шаг играет важнейшую роль в построении модели, поскольку любой неучтенный фактор может оказать существенное влияние на дальнейшее функционирование системы.

Сопоставление найденных неопределенностей с системными для получения вероятностного распределения по модели является заключительным этапом. На основе параметров действий (ресурсов, зависимостей объектов) сначала определяется соответствующий контекстный фактор в контекстной модели. Рассматривая связь неопределенностей и фактора контекста, мы затем выясняем, какие из неопределенностей повлияют на системные действия. Разработка моделей распределения вероятностей требует глубокого анализа (или опыта моделирования) влияния на систему различных комбинаций неопределенностей. Например, известное распределение Дирихле позволяет разработчику уверенно воплощать свои знания в модель. Если нет достаточной априорной уверенности в отношении некоторых факторов, то они будут изучены непосредственно во время работы системы.

Для того чтобы проиллюстрировать сказанное, воспользуемся примером, приведенным в работе [10]. Авторами работы рассматривается тематическое исследование, основанное на новейшей технологии – облачных вычислениях. В статье некоторая компания хочет развернуть свой сайт с помощью облачного сервиса. Сайт будет работать в двух режимах: графическом, который предоставляет больше информации и потребляет больше ресурсов, и текстовом. Назначение сайта заключается в том, чтобы предоставлять пользователю актуальную рыночную информацию с минимальными временными задержками. Владельцы сайта также заинтересованы в том, чтобы снизить затраты на приобретение облачных услуг путем минимизации избыточных в те или иные моменты времени ресурсов.

Очевидно, что такой веб-сайт является самоадаптируемой программной системой, причем наиболее важным фактором при планировании его поведения является время отклика. Анализируя поставленную задачу, авторы работы обнаружили два возможных сценария поведения веб-сайта в случае, когда временная задержка становится недопустимо большой:

1. Выделить больше ресурсов для повышения производительности сайта.

2. Переключиться на текстовый режим, чтобы сократить время вычислений.

Рассмотрим другую ситуацию – уменьшение трафика веб-сайта. Здесь возможны два сценария:

1. Снизить количество неиспользуемых потребляемых ресурсов.

2. Переключиться в графический режим для предоставления пользователю более полной информации.

В соответствии с поставленной задачей и спецификой предметной области были введены следующие системные состояния:

- *Режим = {Графический, текстовый};*
- *Время Отклика = {Нормальное, Низкое};*
- *Ресурсы = {Мало, Средне, Много}.*

Для смены состояний были предусмотрены следующие действия:

- *Добавить или Освободить ресурс;*
- *Переключиться в Графический* либо *Текстовый режим.*

Функция вознаграждения в данном случае связана с временем отклика сайта. Другими побочными факторами, связанными с вознаграждением, будут: режим работы (графический предпочтительнее), количество ресурсов (малое количество предпочтительнее).

Следующим шагом является моделирование неопределенностей. Это предметно-ориентированный процесс, требующий экспертного анализа. В данном случае для простоты была смоделирована только одна неопределенность – доступность облачного ресурса. Поскольку ресурсы ограничены, может возникнуть ситуация, когда провайдер не сможет предоставить их достаточное количество. Вероятность возникновения такой ситуации оценивается в 2 %, а следствием ее может быть задержка в смене состояний системы, либо незапланированная смена состояний. Сопоставляя данную неопределенность с действиями самой системы, авторы работы обнаружили, что она будет связана с действием *Добавить больше ресурсов*. Следовательно, вероятностное распределение было связано с переходами между состояниями, вызванными данным действием. Также для простоты была полностью проигнорирована внутренняя неопределенность системы и сделано допущение, что все другие переходы между состояниями будут совершаться с вероятностью, равной единице.

Диаграмма переходов между состояниями облачного самоадаптируемого веб-сервиса приведена на рис. 2.

Заключение

К настоящему времени в сфере технологий разработки программного обеспечения удалось достичь значительных успехов. Проблема создания интеллектуализированного (и тем самым более надежного) ПО становится все более актуальной [11]. Основными результатами данной работы являются анализ существующих методов создания самоадаптивных программных систем и определение их основных недостатков; выявление преимуществ использования метода машинного обучения с подкреплением для реализации самоадаптивного поведения программы, а также преимуществ модифицированной модельно-ориентированной процедуры машинного обучения, позволяющей учитывать специфику предметной области программной системы и возникающие в процессе эксплуатации неопределенности.

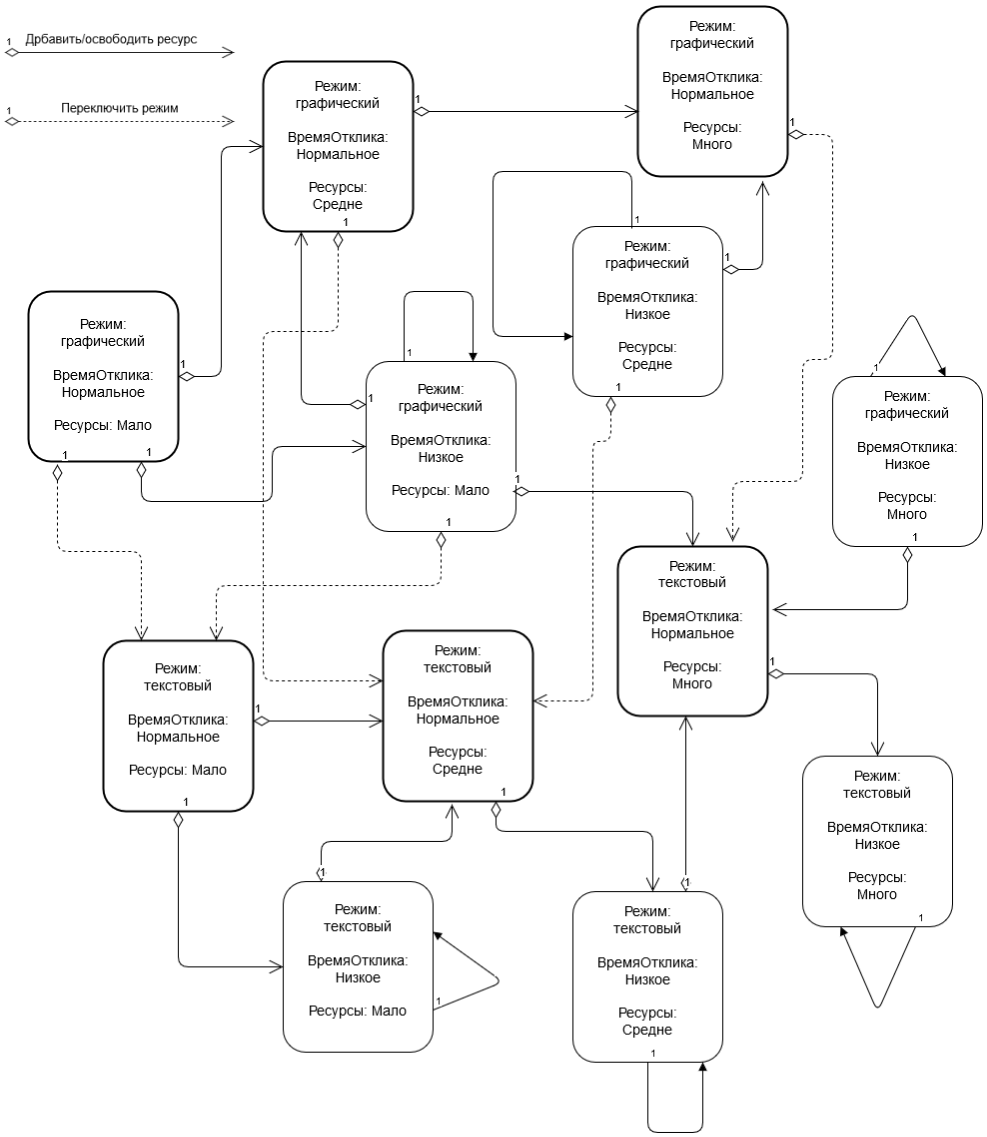

Рис. 2. Диаграмма переходов между состояниями
облачного самоадаптируемого веб-сервиса

Библиографический список

1. **Ghezzi, C.** ContextErlang:A language for distributed context-aware self-adaptive applications / C. Ghezzi, G. Salvaneschi, M. Pradella // Science of Computer Programming. – Amsterdam : Elsevier, 2015. – С. 20–43.
2. **Lei, Y.** A model driven agent-oriented self-adaptive software development method / Y. Lei, K. Ben, Z. He // 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD). – New Jersey : IEEE, 2015. – С. 255–265.
3. **Li, Y.** ADAPT: An agent-based development toolkit and operation platform for self-adaptive systems / Y. Li, L. Li, W. Wang, T. Wu // IEEE Conference on Open Systems (ICOS). – New Jersey : IEEE, 2017. – С. 145–167.
4. **Bencomo, N.** Supporting decision-making for self-adaptive systems: From goal models to dynamic decision network / N. Bencomo, A. Belaggoun // International

Working Conference on Requirements Engineering: Foundation for Software Quality. – San Francisco : IEEE , 2013. – С. 221–237.
5. **Belhaj, N.** Framework for Building Self-Adaptive Component Applications Based on Reinforcement Learning. Framework for Building Self-Adaptive Component Applications Based on Reinforcement Learning / N. Belhaj, , D. Belaid, H. Mukhtar // 2018 IEEE International Conference on Services Computing (SCC). – San Francisco : IEEE, 2018. – С. 139–156.
6. **Dongsun, K.** Reinforcement Learning-Based Dynamic Adaptation Planning Method for Architecture-based Self-Managed Software / K. Dongsun, P. Sooyong // 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. – San Francisco : IEEE, 2009. – С. 178–184.
7. **Colette, R.** Guiding Goal Modeling Using Scenarios / R. Colette, S. Carine, B. Camille // IEEE Transactions on Software Engineering. – San Francisco : IEEE, 1998. – С. 114–127.
8. **Naeem, E.** Uncertainty in Self-Adaptive Software Systems / E. Naeem, M. Sam // Uncertainty inSelf-Adaptive Software Systems. – Berlin : Springer, 2013. – С. 214–238.
9. **Villegas, N. M.** ManagingDynamic Context to Optimize Smart Interactions and Services / Norha M. Villegas, Hausi A. Muller // The Smart Internet. – Berlin : Springer, 2010. – С. 289–318.
10. **Han, H.** Model-based Reinforcement Learning Approach for Planning in Self-Adaptive System / H. Han // International Conference on Ubiquitous Information Management and Communication. – New Jersey : IEEE, 2015. – С. 156–178.
11. **Бождай, А. С.** Метод рефлексивной самоадаптации программных систем / А. С. Бождай, Ю. И. Евсеева // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2018. – № 2 (46). – С. 74–86.

References

1. Ghezzi C., Salvaneschi G., Pradella M. *Science of Computer Programming*. Amsterdam: Elsevier, 2015, pp. 20–43.
2. Lei Y., Ben K., He Z. *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*. New Jersey: IEEE, 2015, pp. 255–265.
3. Li Y., Li L., Wang W., Wu T. *IEEE Conference on Open Systems (ICOS)*. New Jersey: IEEE, 2017, pp. 145–167.
4. Bencomo N., Belaggoun A. *International Working Conference on Requirements Engineering: Foundation for Software Quality*. San Francisco: IEEE, 2013, pp. 221–237.
5. Belhaj N., Belaid D., Mukhtar H. *2018 IEEE International Conference on Services Computing (SCC)*. San Francisco: IEEE, 2018, pp. 139–156.
6. Dongsun K., Sooyong P. *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. San Francisco: IEEE, 2009, pp. 178–184.
7. Colette R., Carine S., Camille B. *IEEE Transactions on Software Engineering*. San Francisco: IEEE, 1998, pp. 114–127.
8. Naeem E., Sam M. *Uncertainty inSelf-Adaptive Software Systems*. Berlin: Springer, 2013, pp. 214–238.
9. Villegas N. M., Muller H. A. *The Smart Internet*. Berlin: Springer, 2010, pp. 289–318.
10. Han H. *International Conference on Ubiquitous Information Management and Communication*. New Jersey: IEEE, 2015, pp. 156–178.
11. Bozhday A. S., Evseeva Yu. I. *Izvestiya vysshikh uchebnykh zavedeniy. Povolzhskiy region. Tekhnicheskie nauki* [University proceedings. Volga region. Engineering sciences]. 2018, no. 2 (46), pp. 74–86. [In Russian]

Бождай Александр Сергеевич
доктор технических наук, профессор, кафедра систем автоматизированного проектирования, Пензенский государственный университет (Россия, г. Пенза, ул. Красная, 40)

E-mail: bozhday@yandex.ru

Bozhday Aleksandr Sergeevich
Doctor of engineering sciences, professor, sub-department of CAD systems, Penza State University (40 Krasnaya street, Penza, Russia)

Евсеева Юлия Игоревна
кандидат технических наук, доцент, кафедра систем автоматизированного проектирования, Пензенский государственный университет (Россия, г. Пенза, ул. Красная, 40)

E-mail: shymoda@mail.ru

Evseeva Yuliya Igorevna
Candidate of engineering sciences, associate professor, sub-department of CAD systems, Penza State University (40 Krasnaya street, Penza, Russia)

Артамонов Дмитрий Владимирович
доктор технических наук, профессор, первый проректор Пензенского государственного университета (Россия, г. Пенза, ул. Красная, 40)

E-mail: aius@pnzgu.ru

Artamonov Dmitriy Vladimirovich
Doctor of engineering sciences, professor, First Vice Rector of Penza State University (40 Krasnaya street, Penza, Russia)

Образец цитирования:

Бождай, А. С. Использование машинного обучения с подкреплением в создании самоадаптивного программного обеспечения / А. С. Бождай, Ю. И. Евсеева, Д. В. Артамонов // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2019. – № 3 (51). – С. 58–68. – DOI 10.21685/2072-3059-2019-3-5.