УДК 004.5
DOI: 10.15827/0236-235X.116.023-026
Дата подачи статьи: 07.10.16
2016. Т. 29. № 4. С. 23–26

АРХИТЕКТУРА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА ДЛЯ ВЗАИМОДЕЙСТВИЯ С МУЛЬТИАГЕНТНОЙ СРЕДОЙ

И.А. Сидоров, к.т.н., научный сотрудник, ivan.sidorov@icc.ru
(Институт динамики систем и теории управления СО РАН, ул. Лермонтова, 134, г. Иркутск, 664033, Россия)

В работе рассматривается подход к созданию пользовательского интерфейса для взаимодействия с мультиагентной средой, ориентированной на решение ресурсоемких вычислительных задач и объединяющей множество разнородных программно-аппаратных ресурсов. Наличие большого числа пользователей в такой среде и разнообразие спектра решаемых ими задач обусловливают необходимость предоставления уровней обслуживания, соответствующих разным категориям пользователей и классам задач. На сегодняшний день наблюдается дефицит высокоуровневых инструментальных средств, позволяющих создавать и применять такого рода мультиагентные среды на основе агентов, представленных в виде сервисов, с ориентацией на некоторую предметную область. Кроме того, в существующих инструментариях мало внимания уделяется интерфейсу, с помощью которого различные категории пользователей (администраторы, разработчики, конечные пользователи) могут взаимодействовать с мультиагентной средой. В данной статье предлагается оригинальная архитектура пользовательского веб-интерфейса мультиагентной среды, основанная на перспективной концепции Agent-as-a-Service, для реализации взаимодействия между пользователями и агентами среды. Предлагаемая архитектура базируется на принципе построения толстого клиента, при котором все основные компоненты веб-приложения загружаются в веб-браузер пользователя и дальнейшая генерация веб-страниц осуществляется динамически на стороне клиента. Архитектурный каркас веб-приложения, функционирующего в веб-браузере пользователя, спроектирован с применением схемы MVC (модель-представление-контроллер), при которой модель данных веб-приложения, пользовательский интерфейс (представление) и логика работы (контроллер) разделены на три слабозависимых компонента, что позволяет выполнять модификацию какого-либо из компонентов с минимальным воздействием на другие. Представленный в статье подход к построению подсистемы пользовательского веб-интерфейса обладает рядом преимуществ: снижает нагрузку на узел агента, повышает время отклика веб-интерфейса, улучшает кросс-платформенность и встраиваемость компонентов пользовательского интерфейса в различные подсистемы мультиагентной среды, облегчает тестирование и поддержку веб-приложения, а также решает ряд проблем с обеспечением безопасности.

Ключевые слова: *мультиагентные технологии, архитектура веб-приложений, визуальный пользовательский интерфейс.*

В настоящее время все большую актуальность приобретают исследования, связанные с повышением интеллектуализации систем организации *распределенных вычислительных сред* (РВС) [1] на основе мультиагентных технологий. Актуальность таких исследований обусловлена необходимостью более эффективного интегрированного использования разнородных ресурсов РВС, а также высокоуровневой поддержки пользователей в процессе создания и запуска масштабируемых приложений в РВС. Пользователи являются, как правило, специалистами в разных предметных областях. Каждый пользователь предъявляет свои субъективные требования к выполнению приложений и использованию общих ресурсов среды. Наличие большого числа пользователей РВС и разнообразие спектра решаемых ими задач обусловливают необходимость предоставления уровней обслуживания, соответствующих разным категориям пользователей и классам задач. Одним из подходов для решения описанных задач является использование *мультиагентных систем* (МАС) с экономическими механизмами управления распределенными вычислениями, обеспечивающее повышение эффективности вычислительных процессов для пользователей в зависимости от степени их заинтересованности в таком повышении [2]. Однако на сегодняшний день наблюдается дефицит высокоуровневых инструментальных средств, позволяющих создавать и применять МАС на основе агентов, представленных в виде сервисов, с ориентацией на некоторую предметную область [3]. Кроме того, в существующих инструментариях мало внимания уделяется интерфейсу, с помощью которого различные категории пользователей (администраторы, разработчики, конечные пользователи) могут взаимодействовать с мультиагентной средой.

При проектировании и реализации пользовательских интерфейсов применяется широкий спектр различных архитектурных подходов (MVC, MVP, MVVM, MDA и др.), используются различные инструментарии и технологии для создания как серверной части веб-приложения (JavaEE, Node.js, Yii, Django и др.), так и клиентской части (jQuery, AngularJS, ReactJS и др.). При реализации современных веб-приложений все больше проявляются тенденции к применению сервис-ориентированного подхода [4] для организации базовой части веб-приложения в виде веб-сервиса. Однако на сегодняшний день недостаточно развиты средства для применения архитектурных подходов SOA и ROA [5] к интеграции разнородных компонентов веб-приложения. Расширение возможностей применения приведенных подходов, с точки зрения

автора, позволит выполнять безболезненную модификацию и расширение набора компонентов веб-приложения, взаимодействующих на основе протокола SOAP либо архитектурного стиля RESTful [6], осуществлять динамическую реконфигурацию и адаптацию пользовательского интерфейса для различных категорий пользователей, расширять функции пользовательского интерфейса администратором РВС с гарантированным уровнем безопасности и надежности системы.

В данной статье предлагается оригинальная архитектура пользовательского веб-интерфейса для взаимодействия с агентами МАС, основанная на сервис-ориентированном подходе. Эта архитектура включает компоненты взаимодействия с разработчиками, администраторами и конечными пользователями МАС. Ее отличительной особенностью является использование перспективной концепции Agent-as-a-Service для взаимодействия между пользователями и агентами МАС.

Подсистема пользовательского веб-интерфейса МАС должна удовлетворять следующим исходным требованиям:

– являться легковесной автономной подсистемой, способной к встраиванию в различные компоненты мультиагентной среды;

– использовать архитектурные подходы SOA и ROA к интеграции разнородных компонентов веб-приложения и подсистем агента;

– обеспечивать независимость от ресурсоемких СУБД, системных сервисов и служб;

– поддерживать проблемную ориентацию создаваемой МАС;

– поддерживать все современные операционные системы для серверной части веб-приложения и все современные веб-браузеры для клиентской части веб-приложения;

– удовлетворять высоким требованиям обеспечения безопасности МАС;

– использовать отечественные и/или свободно распространяемые зарубежные библиотеки и средства создания пользовательского веб-интерфейса;

– предоставлять высокий уровень расширяемости веб-интерфейса с возможностью адаптации для различных нужд МАС;

– предоставлять пользователю информационно-справочные материалы с базовыми функциями экспертной подсистемы для разрешения проблемных ситуаций.

С целью создания подсистемы пользовательского веб-интерфейса, удовлетворяющей приведенным требованиям, была спроектирована и реализована архитектура (см. рисунок), включающая следующие основные составляющие:

– уровень пользователей (разработчики, пользователи и администраторы МАС) и сторонних программных систем, взаимодействующих с агентами МАС по прикладному программному интерфейсу;

– веб-приложение, загружаемое в веб-браузер пользователя и содержащее в своем составе весь необходимый перечень визуальных и графических элементов, моделей данных, декларативных описаний отображения графических элементов и данных модели, а также функциональной составляющей, определяющей логику работы веб-приложения и взаимодействия с агентом МАС;

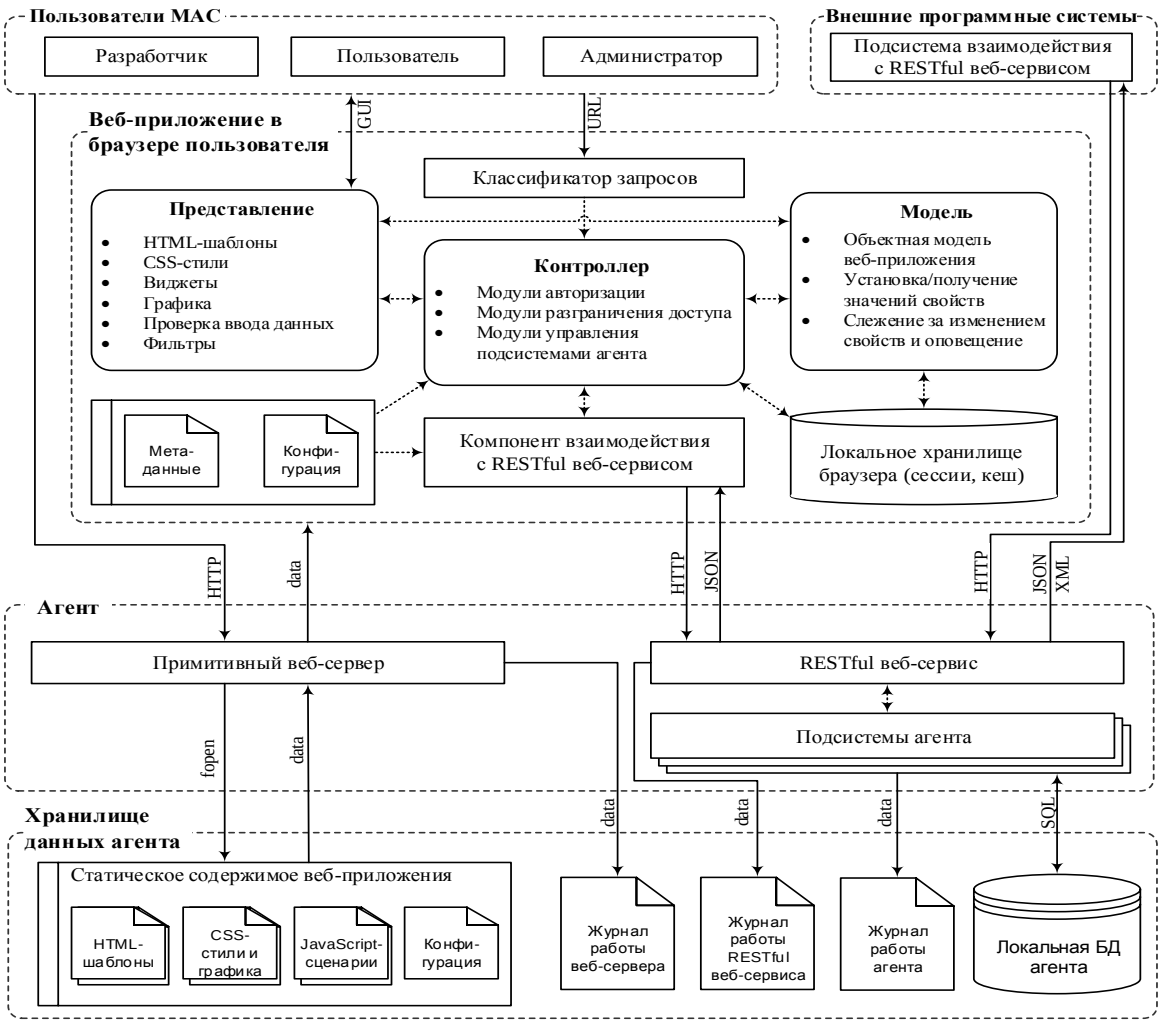
– примитивный веб-сервер, либо реализуемый в составе агента МАС, либо запускаемый независимо и выполняющий единственную задачу по передаче статических файлов веб-приложения в веб-браузер пользователя при первой загрузке;

– RESTful веб-сервис, реализуемый в составе агента и предоставляющий прикладной программный интерфейс для взаимодействия веб-приложения с подсистемами агента на основе протокола HTTP.

Приведенная архитектура базируется на принципе построения толстого клиента, при котором генерация веб-страниц осуществляется динамически на стороне веб-браузера. На сервере хранится статическая копия исходных файлов веб-приложения, которые передаются в веб-браузер при первом обращении пользователя и при дальнейшей работе пользователя с веб-приложением не обновляются. При этом единственной функцией веб-сервера является выполнение GET-запросов для передачи статических файлов по запросу веб-браузера. Реализация такого примитивного веб-сервера (с учетом соблюдения необходимых требований безопасности) составляет не более 150–200 строк программного кода на таких языках программирования, как Java, PHP, C++ (с использованием библиотек Boost или Qt), что позволяет осуществлять реализацию веб-сервера в составе агента. Кроме того, возможно использование существующих легковесных программных решений, таких как Mongoose [7], node.js [8], а также более комплексных программных продуктов Apache или nginx, однако применительно к поставленным задачам выбор таких решений не является рациональным.

Архитектурный каркас веб-приложения, функционирующего в веб-браузере пользователя, спроектирован с применением схемы MVC [9] (модель-представление-контроллер), при которой модель данных веб-приложения, пользовательский интерфейс (представление) и логика работы (контроллер) разделены на три слабозависимых компонента, что позволяет выполнять модификацию какого-либо из компонентов с минимальным воздействием на другие. Отдельные триплеты модели, представления и контроллера, выполняющие определенные задачи, объединяются в модули (к примеру, модули авторизации, управления настройками агента, управления ресурсами и др.).

При первом открытии в веб-браузер загружается главная страница (index.html), содержащая список ресурсов (HTML, CSS, JavaScript и др.), ко-



Архитектура пользовательского интерфейса для взаимодействия с агентом МАС

The architecture of the user interface to interact with MAS agent

торые должны быть загружены дополнительно для инициализации и дальнейшей полноценной работы веб-приложения. Кроме того, главная страница содержит базовую структуру элементов пользовательского интерфейса и является оболочкой для всех отображаемых в дальнейшем веб-страниц, наполняемых данными требуемым контроллером на основе необходимого HTML-шаблона. Эти шаблоны содержат декларативное описание размещения графических и текстовых элементов с двусторонним связыванием с моделью данных.

При переходе по веб-страницам (по всему веб-приложению) не производится загрузка новых веб-страниц с веб-сервера, а загружаются только те данные, которые необходимо отобразить пользователю. Выявление переходов между веб-страницами выполняется классификатором запросов, который определяет действие пользователя и передает управление необходимому контроллеру. Если осуществляется переход на страницу, на которой должны быть отображены данные, получаемые от подсистем агента, контроллер делает запрос к компоненту взаимодействия с RESTful веб-сервисом агента, который, в свою очередь, формирует и выполняет в асинхронном режиме Ajax-запрос по протоколу HTTP к RESTful веб-сервису агента. После получения ответа (в качестве формата передачи данных используется JSON) осуществляются проверка полученных данных, изменение значений параметров модели и дальнейшее отображение этих данных пользователю в составе необходимого HTML-шаблона.

В составе рассматриваемого пользовательского веб-интерфейса на текущий момент частично реализованы следующие модули:

– аутентификация и авторизация пользователей для выполнения дальнейших операций на основе выданного уникального ключа (token);

– взаимодействие с агентом: получение списка заданий и подробной информации об отдельном задании, добавление в очередь нового задания с указанием параметров запуска, изменение параметров исполняемого задания, удаление задания из очереди;

– выполнение операций с ресурсами агента (загрузка, получение, изменение и удаление файлов заданий);

– изменение настроек агента (изменение квот, политик управления заданиями и др.).

Для указанных операций используются четыре основных метода CRUD (Create, Retrieve, Update, Delete), реализуемых при построении RESTful веб-сервисов в виде методов HTTP-протокола: POST, GET, PUT и DELETE. При реализации описанного подхода использованы следующие средства и библиотеки: HTML5, CSS3, JavaScript, jQuery, Bootstrap, модифицированная версия Angular 2 [10].

Представленный в статье подход к построению подсистемы пользовательского веб-интерфейса обладает рядом преимуществ: снижает нагрузку на веб-сервер, функционирующий на узле агента, за счет вынесения логики приложения и функций генерации веб-страниц на сторону веб-браузера пользователя; повышает время отклика веб-интерфейса (передаются только данные); улучшает кросс-платформенность и встраиваемость компонентов пользовательского интерфейса в различные подсистемы мультиагентной среды; облегчает тестирование и поддержку веб-приложения, а также решает ряд проблем с обеспечением безопасности.

Исследование выполнено при частичной финансовой поддержке РФФИ, проекты №№ 15-29-07955-офи_м и 16-07-00931, а также при частичной финансовой поддержке Совета по грантам Президента Российской Федерации для государственной поддержки ведущих научных школ Российской Федерации НШ-8081.2016.9.

Литература

1. Гергель В.П., Линев А.В. Проблемы и перспективы достижения экзафлопного уровня производительности суперкомпьютерных систем // Вестн. Нижегород. ун-та им. Н.И. Лобачевского. 2012. № 3–1. С. 189–198.

2. Toporkov V.V., Emelyanov D.M. Economic model of scheduling and fair resource sharing in distributed computations. Programming and Comp. Soft., 2014, vol. 40, no. 1, pp. 35–42.

3. Бычков И.В., Опарин Г.А., Феоктистов А.Г., Сидоров И.А., Богданова В.Г., Горский С.А. Мультиагентное управление вычислительной системой на основе метамониторинга и имитационного моделирования // Автометрия. 2016. Т. 52. № 2. С. 3–9.

4. Buyya R., Vecchiola C., Selvi S.T. Mastering cloud computing. Burlington, Massachusetts, USA, Morgan Kaufmann Publ., 2013, 469 p.

5. Guo X., Shen J., Yin Z. On software development based on SOA and ROA. Control and Decision Conf., Publishing Press, 2010, pp. 1032–1035.

6. Pautasso C., Zimmermann O., Leymann F. RESTful web services vs. big web services: making the right architectural decision. Proc. 17th Intern. World Wide Web Conf., Beijing, China, 2008, pp. 805–814.

7. Mongoose Embedded Web Server Library. URL: https://github.com/cesanta/mongoose (дата обращения: 06.10.2016).

8. Gackenheimer C. Node.js recipes: a problem-solution approach, Apress, 2013, 376 p.

9. Scott E.A. SPA design and architecture: understanding single page web applications, Manning Publ., 2015, 275 p.

10. Deeleman P. Learning Angular 2, Packt Publ., 2016, 354 p.

Software & Systems Received 07.10.16
DOI: 10.15827/0236-235X.116.023-026 2016, vol. 29, no. 4, pp. 23–26

USER INTERFACE ARCHITECTURE FOR INTERACTION WITH MULTI-AGENT ENVIRONMENT

I.A. Sidorov[1], *Ph.D. (Engineering), Research Associate, ivan.sidorov@icc.ru*
[1] *Institute of System Dynamics and Control Theory SB RAS, Lermontov St. 134, Irkutsk, 664033, Russian Federation*

Abstract. The paper discusses an approach to the development of user interface to interact with a multi-agent environment, which is intended to solve resource-intensive tasks and includes a lot of loosely-coupled multivendor computing resources. Nowadays, there is a lack of toolkits which allow creating and using this kind of multi-agent environment, with a focus on a certain subject area. In addition, in the existing toolkits they do not pay enough attention to a graphical user interface, in which different categories of users (administrators, developers, and end users) can interact with the multi-agent environment. This paper proposes an original architecture of the user interface based on the promising concept of Agent-as-a-Service to implement the interaction between users and agents of the environment. The proposed architecture is based on the principle of building a thick client, in which components of a web-application are loaded into user's web-browser, and further web pages generation is performed dynamically on the client side. The architecture of a web application is designed using MVC scheme (model-view-controller), where a web-application model of data, user interface (view) and logic (controller) are divided into three weakly dependent components that allow modifying any component with minimal effect on the other. The presented approach to the development of the user web interface has several advantages: it reduces the load on an agent's node, increases the response time of a web interface, enhances portability and embeddability of user interface components in various subsystems of a multi-agent environment, facilitates testing and support of web applications, and solves a number of safety problems.

Keywords: multi-agent technologies, architecture of web applications, graphical user interface.

Acknowledgements*. The research was supported by RFBR, projects no. 15-29-07955-ofi m and no. 16-07-00931-a, and partially supported by the Council for Grants of the President of the Russian Federation for state support of the leading scientific schools, project NSh-8081.2016.9.*

References

1. Gergel V.P., Linev A.V. Problems and prospects for exaflopx performance level in supercomputer systems. *Vestnik Nizhegorodskogo Universiteta* [Bulletin of Lobachevsky Univ. of Nizhni Novgorod]. 2012, vol. 3, no. 1, pp. 189–198 (in Russ.).

2. Toporkov V., Emelyanov D. Economic Model of Scheduling and Fair Resource Sharing in Distributed Computations. *Programming and Computer Software*. 2014, vol. 40, no. 1, pp. 35–42.

3. Bychkov I.V., Oparin G.A., Feoktistov A.G., Sidorov I.A., Bogdanova V.G., Gorsky S.A. Multi-agent Control of Computational Systems on the Basis of Meta-monitoring and Imitational Simulation. *Avtometriya* [Optoelectronics, Instrumentation and Data Processing]. Allerton Press, 2016, vol. 52, no. 2, pp. 107–112 (in Russ.).

4. Buyya R., Vecchiola C., Selvi S.T. Mastering Cloud Computing, Burlington, Massachusetts, USA, Morgan Kaufmann Publ., 2013, 469 p.

5. Guo X., Shen J., Yin Z. On software development based on SOA and ROA. *Control and Decision Conf. (CCDC)*. Publishing Press, 2010, pp. 1032–1035.

6. Pautasso C., Zimmermann O., Leymann F. RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision. *Proc. 17th Int. World Wide Web Conf*. Beijing, China, 2008, pp. 805–814.

7. *Mongoose Embedded Web Server Library*. Aavailable at: https://github.com/cesanta/mongoose (accessed October 6, 2016).

8. Gackenheimer C. *Node.js Recipes: A Problem-Solution Approach*. Apress, 2013, 376 p.

9. Emmit S. *SPA Design and Architecture: Understanding Single Page Web Applications*. Manning Publ., 2015, 275 p.

10. Deeleman P. *Learning Angular 2*. Packt Publ., 2016, 354 p.