Визуальное Программирование Сценариев Системы Управления Интеллектуального Здания

П.Л. Николаев, Н.В. Жига

***Аннотация* — Предложена система управления интеллектуального здания и клиент для него, позволяющий создавать сценарии системы при помощи визуального программирования с использованием браузера. Приведены архитектура системы, архитектура клиента, клиент-серверный способ взаимодействия двух приложений, в котором клиент держит связь с сервером при помощи протокола WebSocket. Отправка данных при взаимодействии приложений осуществляется по протоколу HTTP, в котором передаются данные плана для системы в виде JSON. Предложены два типа редакторов сценариев, управляемых пользователем с помощью браузера при помощи технологий HTML, CSS, ECMAScript. Простая версия редактора создает простой план с одним триггером. Сложная версия редактора позволяет создать более усложненный план, визуализируя его в виде дерева. Также рассмотрен пример создания сложного сценария управления освещенностью помещения с помощью разработанного клиента.**

***Ключевые слова* — Визуальное программирование, интеллектуальное здание, клиентское приложение для системы управления, умный дом.**

I. Введение

Интеллектуальное здание (умный дом) представляет собой систему, объединяющую в себе все системы здания. Она влияет на поведение подсистем здания с помощью установки и считывания показателей различных типов: целых и вещественных чисел, координат двумерного и трехмерного пространств, временных показателей, булевых значений, строковых значений. Подсистемами здания могут выступать различные системы жизнеобеспечения, системы охраны и прочие, так или иначе выполняющие какую-то свою функциональную задачу. Функцией системы интеллектуального здания является слежение за подсистемами здания, создание сценариев, автоматизирующих различные задачи, например жизнеобеспечения. Система интеллектуального здания должна обладать простым интерфейсом управления, низкими затратами на внедрение и сопровождение.

Зачастую, производители и интеграторы систем автоматизации зданий не предоставляют своим клиентам, пользователям таких систем, простых и удобных средств настройки их функционирования. Если пользователю понадобится внести какие-то изменения в работу системы управления умного дома, например, перенастроить автоматическое управление освещением, то из-за отсутствия необходимых знаний ему придется обращаться за технической поддержкой к производителю или интегратору.

В связи со всем вышеизложенным, для наиболее удобной и быстрой работы со сценариями пользователю, не имеющему знаний в области программирования, требуется инструмент, позволяющий создавать и редактировать сценарии в визуальной среде.

Процесс визуального программирования заключается в манипулировании графическими блоками, которыми задаются переменные, логические выражения, циклы, функции и т.п. Данный способ является гораздо более эффективным и понятным для восприятия человека, нежели написание кода.

Визуальное программирование позволяет избежать множества ошибок и значительно сократить время реализации алгоритмов [1]. Оно позволяет создавать и редактировать сценарии интеллектуального здания без привлечения специалистов в области программирования при помощи наглядного и простого интерфейса. Однако для сохранения простоты и ясности интерфейса потребуется использовать его в узко поставленных задачах.

Интерфейс интеллектуального здания должен обладать рядом критериев.

1. Простая настройка сценариев через интерфейс.

Интерфейс должен позволять пользователю настраивать различное поведение системы при возникновении определенных ситуаций, например, включать и поддерживать искусственное освещение внутри здания на определенном уровне. То есть система должна позволять пользователю создавать сценарии. При этом в целях снижения затрат на сопровождение системы пользователю не обязательно обладать компетенциями в области программирования.

2. Доступность интерфейса.

Интерфейс должен быть максимально доступным пользователю, не требовать от пользователя специфических знаний. Пользователь должен легко подключаться к системе наиболее доступным средством. Интерфейс должен быть доступен на большинстве платформ: на персональных компьютерах (Windows, MacOS, Unix-подобные ОС), на смартфонах и планшетах (Android, iOS).

Статья получена 04 октября 2020.

П.Л. Николаев – старший преподаватель Московского авиационного института (национального исследовательского университета) (e-mail: npavel89@gmail.com).

Н.В. Жига – магистрант Московского авиационного института (национального исследовательского университета) (zhiga.nv@ya.ru).

3. Информативность интерфейса.

Интерфейс должен максимально информативно показывать аномалии, возникшие в системе. Пользователь должен сразу же получить информацию о местоположении аномалии для оперативного принятия решения и устранения возникших неполадок.

4. Поддержка существующих систем

При наличии готовой поддержки существующих подсистем интеграция системы умного дома становится тривиальной задачей, что снизит затраты на интеграцию с существующими подсистемами

Цель данной работы — разработать приложение для создания сценариев работы интеллектуального здания, доступное пользователю, не владеющего навыками программирования, с простым в освоении и интуитивно понятным интерфейсом.

Для обеспечения доступности интерфейса для приложения была выбрана платформа Web. Данная платформа позволит обеспечить доступ к интерфейсу с помощью веб-браузера. При этом становится не нужно разрабатывать клиент для каждой операционной системы и устанавливать ее каждому пользователю. Становится возможным централизованное обновление приложения на собственных серверах, становится возможна непрерывная разработка приложения с отсутствием строгого расписания релизов клиентского приложения. Также это позволит исполнять тяжелые вычисления на стороне сервера, что позволит сэкономить на оборудовании клиента. В качестве технологий внутри платформы были выбраны стандартные HTML5, CSS3 и ECMAScript 5.1 [2].

II. Обзор существующих работ

Существует несколько средств визуального программирования, на основе которых возможна реализация собственных специальных решений. К ним можно отнести такие среды, как Scratch [3], Snap! [4] и Blockly [5]. В данных средах создание программ осуществляется путем правильной расстановки блоков перетаскиванием (drag-and-drop) таким образом, чтобы получилась целостная и функционирующая логическая конструкция.

Scratch – это объектно-ориентированная визуальная среда программирования, прежде всего предназначенная для обучения программированию школьников.

Snap! – расширенная реализация среды Scratch. Среда Snap! реализована на языке JavaScript и работает в браузере. Главное отличие от Scratch состоит в возможности добавления собственных компонентов и в сохранении созданных программ в формате XML.

Blockly от Google представляет собой библиотеку, написанную на JavaScript, и предназначенную для встраивания в качестве визуального редактора в веб-приложения, работающие через браузер, и приложения под операционные системы Android и iOS.

Из всего вышеперечисленного можно выделить две среды разработки: Snap! и Blockly. Оба средства вполне можно использовать для разработки сценариев в различных системах автоматизации. Например, в [6] описано применение Snap! в системе домашней автоматизации. В [7] рассмотрена разработка фреймворка визуального программирования алгоритмов функционирования для беспроводных сенсорных сетей, использующихся в системе умного дома. В качестве основы данного фреймворка используется библиотека Blockly.

И Snap!, и Blockly имеют практически схожий функционал. Следует отметить тот факт, что ни в одной рассмотренной среде нет возможности представления написанных программ в формате JSON. В некоторых случаях используется только XML, а в других – универсальные форматы данных вовсе не используются. Также данные среды обладают избыточным функционалом и требуют довольно большой доработки под необходимые нам задачи. В связи с этим, разумным является разработка собственного средства визуального программирования.

III. Архитектура приложения

Архитектура приложения (рис. 1) является клиент-серверной, где на сервере выполняется обработка данных, исполнение планов и оценка состояния здания, а клиент показывает состояние системы и управляет планами.

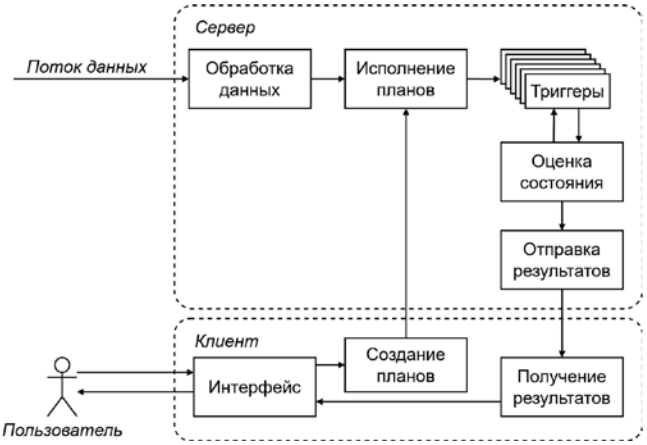


Рис. 1. Архитектура приложения, включающая клиент и сервер.

Рассмотрим клиентскую часть приложения. Пользователь осуществляет взаимодействие с клиентом при помощи веб-браузера по протоколу HTTP. При загрузке клиента устанавливается соединение с серверной стороной по протоколу WebSocket, что позволит пользователю клиента наблюдать за состоянием системы в режиме реального времени. Таким образом, осуществляется “Получение результатов” на схеме.

Пользователь может изменять поведение системы при помощи интерфейса клиента. Тогда сервер отправит специально оформленные данные – планы. В планах указано, какие условия считаются ошибкой, в каких случаях необходимо немедленно сообщить пользователю о текущей ситуации в системе.

Планы способны выполнять автономную работу. Например, при обнаружении сильной задымленности помещения, система способна самостоятельно

предпринять меры устранения – использовать средства тушения пожара.

Процесс сохранения планов показан на рис. 2. Манипуляция с планами через интерфейс реализована двумя способами: первый – простой интерфейс, второй – продвинутый. В простом интерфейсе план изменяется при помощи набора простых правил, простых условий. Если требуется настроить более сложный сценарий, например цепочку условий и реакцию нескольких подсистем на условие, то пользователь может настроить такой сценарий через блок-схемы. При помощи таких блок-схем и строится синтаксическое дерево. В простом интерфейсе неявно используются те же блок-схемы, поэтому в алгоритме построения синтаксического дерева ничего не меняется. Когда дерево построено, происходит проверка дерева на наличие ошибок.

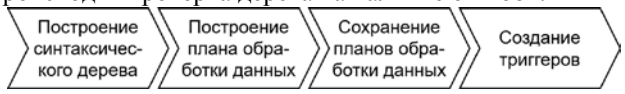


Рис. 2. Процесс сохранения планов приложения.

После проверки дерева на ошибки выполняется построение самого плана. Т.е. данные из дерева строятся уже в таком виде, в котором их удобно передавать серверу по протоколу HTTP, и хранить на самом сервере. Данный протокол работает очень эффективно и может поддерживать множество пользователей [8].

Сервер сохраняет планы обработки данных к себе в хранилище и создает триггеры на поступающие данные. Триггеры – это слушатели потоков данных, реагирующие на изменения в потоке данных. Они включают в себя условие, по которому будут реагировать на изменения данных, и инструкцию, по которой триггеры исполняют автономную работу.

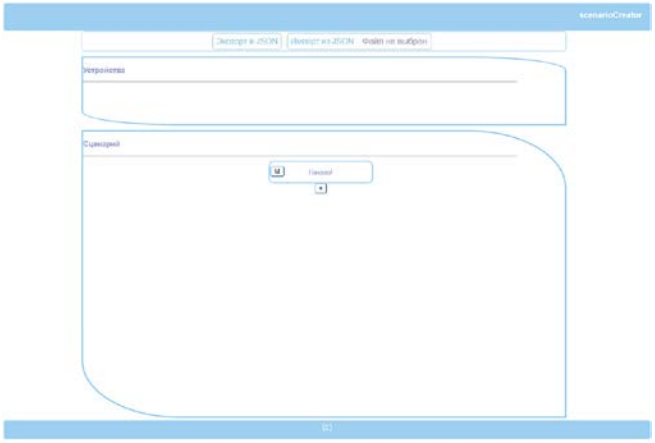
IV. Прототип приложения

Структура веб-страницы:

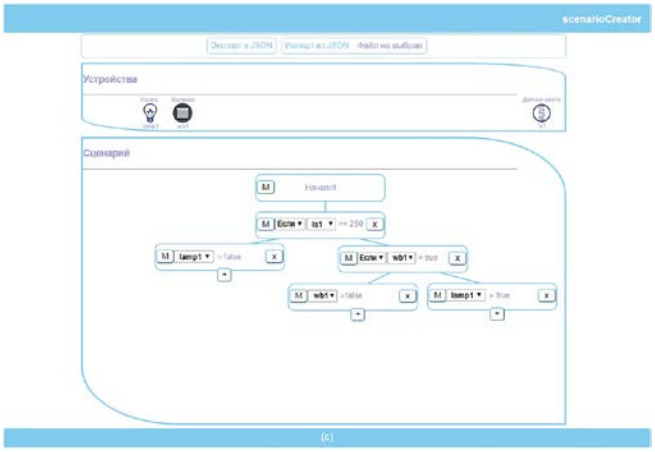
1. Секция статуса системы:
   a) Лента меню;
   b) Блок со списком площадок;
   c) Блок с картой площадки.
2. Секция редактирования:
   a) Лента меню;
   b) Лента инструментов;
   c) Блок со списком планов;
   d) Блок редактора планов.

Секция статуса системы отвечает за показ статуса систем в здании. Если в какой-то части системы здания произошла ошибка, то она отобразится на карте площадки, что будет наглядно демонстрировать, в каком месте произошла ошибка. В секции со списком площадок отображается краткая статистика по ошибкам, например их количество.

Секция редактирования используется для редактирования текущих планов, создания новых и удаления. Блок редактора существует в двух видах: простой и продвинутый (рис. 3). В простом редакторе возможно лишь редактирование (или создание) одного условия и одной реакции на условие. А в продвинутом редакторе возможно редактирование (или создание) множества условий и реакций, объединенных цепочкой.



(а)



(б)

Рис. 3. (а) Продвинутый редактор планов при создании нового плана. (б) Продвинутый редактор планов при редактировании готового плана.

При сохранении планов в редакторе клиентское приложение начинает формирование плана, удобном для передачи серверу по протоколу HTTP в формате JSON. Этот формат отличается своей простотой и универсальностью. Такой формат поддерживается ECMAScript, начиная с версии 3.1 [9]. В этом плане включены такие данные, как имя плана, идентификаторы сенсоров, условия планов, их обработка. Например, существует следующий план: если уличный световой поток, на котором находится сенсор “ls1”, больше или равен 250 лм, то освещать помещение (лампой “lamp1”) не имеет смысла, в противном случае нужно опустить жалюзи (“wb1”) и уже потом только осветить комнату. Такой сценарий приведен на рис. 3б, а его сохраненный план уже в формате JSON – на рис. 4.

```
{
    "name":"Lighting plan",
    "plan":{
        "type":"condition",
        "expression":"if ls1 >= 250",
        "when_true":{
            "type":"act",
            "expression":"lamp1 = false"
        },
        "when_false":{
            "type":"condition",
            "expression":"if wb1 == true",
            "when_true":{
                "type":"act",
                "expression":"lamp1 = true"
            },
            "when_false":{
                "type":"act",
                "expression":"wb1 = false"
            }
        }
    }
}
```


Рис. 4. Пример JSON-схемы, созданный редактором.

V. Заключение

Таким образом, была разработана архитектура приложения для мониторинга интеллектуального здания и создан прототип клиента для этого приложения, который соответствует таким критериям, как простая настройка сценариев через интерфейс, доступность интерфейса, информативность интерфейса. Приложение построено вокруг технологий Web-приложений, как HTML5, CSS3, ECMAScript 5.1, а данные передаются серверу по протоколу HTTP в формате JSON.

Библиография

[1] Николаев П.Л., Хорошко Л.Л. Представление кодированными деревьями сценариев системы управления интеллектуального здания // Системный администратор. 2017. № 173. С. 89-91.

[2] Зеленко О.В., Валеева Л.Р., Климанов С.Г. Обзор современных Web - технологий // Вестник Казанского технологического университета. 2015. т.18, в.2. С. 354-356.

[3] Scratch [Электронный ресурс]. – URL: https://scratch.mit.edu

[4] Snap! [Электронный ресурс]. – URL: http://snap.berkeley.edu

[5] Blockly [Электронный ресурс]. – URL: https://developers.google.com/blockly/

[6] J. Ash et al., “Scratchable Devices: User-Friendly Programming for Household Appliances,” in Proc. Human-Computer Interaction. Towards Mobile and Intelligent Interaction Environments. HCI 2011. Lecture Notes in Computer Science, vol. 6763, 2011, pp. 137–146. DOI: 10.1007/978-3-642-21616-9_16

[7] M.A. Serna, C.J. Sreenan, S. Fedor , “A visual programming framework for wireless sensor networks in smart home applications,” in Proc. 2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Singapore, 2015, pp. 1-6. DOI: 10.1109/ISSNIP.2015.7106946

[8] Фейт С. TCP/IP Архитектура, протоколы, реализация (включая IP версии 6 и IP Security). 2 изд. М.: Лори, 2009. 424 с.

[9] Николас З. ECMAScript 6 для разработчиков. М.: Питер, 2017. 352 с.

Scenarios Visual Programming Of The Intelligent Building Management System

P.L. Nikolaev, N.V. Zhiga

***Abstract* — For the most convenient and fast work with scenarios the user unfamiliar with the field of programming is offered the intelligent building management system and the client for it, which allows to create system scenarios by visual programming using a browser. The system is expected to possess the following criteria: easy scenarios setting up via interface, interface accessibility, interface information content and support for existing systems. The system and client architecture and client-server method of interaction between two applications are presented. The client communicates with the server using WebSocket Protocol and sends data by HTTP Protocol, which transmits plan data for the system in the JSON format. The system is ready to work with central application updating on own servers, the application development without strict scheme of client application releases also becomes possible. Two types of script editors which are can be controlled by user using the browser with HTML, CSS and ECMAScript technologies are suggested. The simple editor version creates a simple plan with a single trigger. The complex version allows to create a more complex plan by its visualization in a tree form. The server saves data processing plans in the storage and creates triggers for incoming data. Triggers include condition of reacting to data changes and instruction for performing off-line work. The example of creation of the complex scenario for controlling room lighting by developed client is given.**

***Keywords* — Client application for system management, intelligent building, smart home, visual programming.**

REFERENCES

[1] Nikolaev, P.L, Khoroshko L.L. (2017) “Predstavleniye kodirovannymi derev'yami stsenariyev sistemy upravleniya intellektual'nogo zdaniya” [Representation of scenarios of intelligent building management system by coded trees], *Sistemnyy Administrator* [System Administrator], 173, pp. 89-91.

[2] Zelenko, O.V., Valeeva, L.R., Klimanov, S.G. (2015) “Obzor sovremennykh Web – tekhnologiy” [Overview of modern Web technologies*], *Vestnik Kazanskogo tekhnologicheskogo universiteta* [Kazan Technological University Bulletin*], 18(2), pp. 354-356.

[3] Scratch , Available at: https://scratch.mit.edu

[4] Snap!, Available at: https://scratch.mit.edu

[5] Blockly, Available at: https://developers.google.com/blockly/

[6] J. Ash et al., “Scratchable Devices: User-Friendly Programming for Household Appliances,” in Proc. Human-Computer Interaction. Towards Mobile and Intelligent Interaction Environments. HCI 2011. Lecture Notes in Computer Science, vol. 6763, 2011, pp. 137–146. DOI: 10.1007/978-3-642-21616-9_16

[7] M.A. Serna, C.J. Sreenan, S. Fedor , “A visual programming framework for wireless sensor networks in smart home applications,” in Proc. 2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Singapore, 2015, pp. 1-6. DOI: 10.1109/ISSNIP.2015.7106946

[8] Feit, S (1999) *TCP/IP: Architecture, Protocols, and Implementation with IPv6 and IP Security* , 2nd edn., New York: Mcgraw-hill.

[9] Nicholas C. Zakas (2016) *Understanding ECMAScript 6: The Definitive Guide for JavaScript Developers* , San Francisco: No Starch Press.