МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МАТЕРИАЛЫ

VI Международной молодежной научной конференции

«МАТЕМАТИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННЫХ, ТЕХНИЧЕСКИХ И ЭКОНОМИЧЕСКИХ СИСТЕМ»

Томск, 24–26 мая 2018 г.

Под общей редакцией
кандидата технических наук И.С. Шмырина

Томск
Издательский Дом Томского государственного университета
2018

тических переменных, что усложняет процесс фаззификации данных, в отличие от стандартного представления входных параметров в числовом виде. Сформированное дерево используется для дальнейшего построения нейро-нечеткого классификатора как системы нечеткого вывода для решения аналитических, научно-технологических задач [3].

ЛИТЕРАТУРА

1. *Горбачев С.В., Сырямкин В.И.* Нейро-нечеткие методы в интеллектуальных системах обработки и анализа многомерной информации. – Томск: Изд-во Томского государственного университета, 2014. – 510 с.

2. *Butkievicz B.S.* Robust fuzzy clustering with fuzzy data // Lecture News in Computer Science. – V. 3528. – Berlin – Heidelberg: Springer-Verlag, 2005. – P.76–82.

3. *Горбачев С.В., Горбачева Н.Н.* Логически прозрачный нейро-нечеткий классификатор для многокритериальной оценки эффективности инновационного развития // Материалы VII Всероссийской научно-практической конференции с международным участием «Региональные проблемы преобразования экономики: интеграционные процессы и социально-экономическая политика региона», Махачкала, 9–10 ноября 2016 г., с. 84–92.

РАЗРАБОТКА USE-ИНСТРУМЕНТА ДЛЯ СОЗДАНИЯ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

К.Н. Попеляева, В.В. Томков
Томский государственный университет
ksyufkas@gmail.com, isirbis@gmail.com

Введение

В настоящее время программисты стремятся упростить процесс разработки программных продуктов различными способами: техническими и управленческими. К техническим можно отнести фреймворки, наборы библиотек для решения часто возникающих задач и т.д. Среди управленческих – методики организации проведения работ на всех этапах создания программного обеспечения, например, Agile, RUP и др. [1].

Иногда часть работ по добавлению или изменению существующего функционала можно переложить на конечных пользователей программного продукта. Так для этого используются различные инструменты для визуального программирования. Такие инструменты позволяют разрабатывать программы без углубленных знаний языков программирования.

1. Визуальное программирование. Проблемы и минимизация рисков

С развитием технологий программные продукты всё больше стали применять для автоматизации бизнес-процессов, протекающих в различных предприятиях. Часть из этого программного обеспечения практически не изменяется в процессе жизни всего предприятия, а некоторая меняется очень часто.

В связи с увеличением потребностей у пользователей, приходится вкладывать всё больше ресурсов, чтобы программное обеспечение оставалось актуальным. Для этого используют технические и управленческие подходы. При использовании технических подходов очень много ресурсов уходит для содержания большого штата разработчиков. При управленческих возможно снизить расход ресурсов на разработку и поддержание программного обеспечения благодаря оптимизации того или иного рода.

Одной из таких оптимизаций является привлечение конечных пользователей программного продукта и небольшого повышения их квалификации. Для этого используются визуальные инструменты программирования, которые имеют ряд проблем.

Первая из них – это нерасширяемость, которая заключается в том, что пользователю предоставляется ограниченный набор методов для решения узкоспециализированных задач [2,3], которые успешно решаются, но при возникновении краевых случаев решение получается в лучшем случае крайне сложным, а в худшем – невозможным.

Вторая связана с генерацией кода, который будет выполнятся. Как правило, он получается медленным в выполнении и сложным в оптимизации. Связано это с тем, что пользователям инструмента предлагают оперировать нечёткими абстракциями.

Третья заключается в бедности и запертости в них. Программное обеспечение, которое предоставляет бедный набор инструментов визуального программирования, например IDE для работы с GUI, ограничивает возможности разработчика при решении задач и буквально запирает его в себе.

Озвученные проблемы возникают перед разработчиками при создании инструментов, которые должны решать задачи без чётко определенных границ [4].

Универсального решения всех озвученных проблем нет, но существует возможность снизить фактор риска провала инструмента еще на стадии его проектирования. Для этого предлагается придерживаться следующего набора правил:

– ограничивать спектр задач минимально, не сводя инструмент к решению узкоспециализированных задач, например, формирование SQL запроса, создание схемы печатной платы и т.д.;
– наполнять инструмент возможностями для работы с задачами, которые возникают в предметной области инструмента;
– оперировать элементарными абстракциями и связями между ними.

Для проверки этих правил на практике было принято решение о создании инструмента для визуального программирования, с помощью которого возможно создание web-приложения с использованием обработчиков событий пользователя.

2. Концепция визуального инструмента программирования

Первоначально нужно определиться с набором элементарных абстракций, с помощью которых пользователь сможет формировать интерфейс. Для этого за основу взяты HTML-элементы, которые можно сгруппировать следующем образом:

– текстовые блоки;
– списки;
– изображения;
– таблицы;
– элементы формы.

Каждая группа содержит в себе набор элементарных абстракций, таких как поле ввода, элемент списка и т.д. Каждый элемент – это объект, имеющий набор специфических параметров – свойств, которые его описывают, а также обязательно содержащий методы сериализации и десериализации, использующиеся для его представления при хранении и получения в БД.

Для реализации композиции элементов могут использоваться параметры: перечень дочерних элементов и родительский элемент.

Предложенная концепция позволяет в дальнейшем расширять набор элементов, которые можно будет использовать при разработке интерфейса и покрывать больший спектр задач, которые будут стоять перед пользователем данного инструмента, что снижает риск возникновения проблемы расширяемости инструмента и позволяет охватывать смежные области.

Следующий этап – формирование концепции создания или изменения пользователем функционала приложения, без необходимости использования языка программирования. Для этого создаются абстракции: «функциональный блок» и «связь». С их помощью можно реализовать концепцию блочного программирования, с небольшими изменениями – «функциональный блок» может содержать в себе набор входных и выходных параметров, а также источник и тип события, на которое он должен быть выполнен.

Таким образом, можно запрограммировать элемент use-case'а реализуемого функционала, что, в общем, позволяет реализовать необходимое функциональное требование, предъявляемое к программному продукту.

Этот подход позволит в дальнейшем генерировать код, который будет выполняться, наименее избыточным и легким для изменений и оптимизации.

Последней задачей, которая возникает при создании инструмента – хранение и генерация исполняемого кода. Для хранения будет использоваться формат JSON, т.к. он позволяет проводить сериализацию и десериализацию как простых, так и сложных структур (рис. 1), что является важным фактором, даже не смотря на использование элементарных абстракций, т.к. даже самый элементарный элемент может в себе содержать множество свойств и их композицию.

```
"WidgetButton" : {
        "id" : "1222",
        "position" : {
                "x" : 100,
                "y" : 100
        },
        "size" : {
                "width" : {
                        "default" : 100,
                        "min" : 50,
                        "max" : 150
                },
                "height" : {
                        "default" : 50,
                        "min" : 30,
                        "max" : 75
                }
        },
        "pressed" : false,
        "relations" : {
                "parentId" : "1210",
                "childrens" : [
                        "1223",
                        "1224"
                ]
        }
}
```

Рис. 1. Структура элемента в формате JSON

Перед тем, как сохранить элемент в БД, у каждого элемента, а также у всех его дочерних элементов, вызывается метод его сериализации. При получении из БД используется обратная операция – десериализация; происходит она аналогично, т.е. после десериализации одного элемента находятся его дочерние элементы и происходит вызов соответствующего метода у них.

После этого этапа получаем их в том виде, которым удобно оперировать для формирования исполняемого кода. Эта часть этапа разбита на две части:

– формирование HTML-элементов, которые будут формировать HTML-документ;

– формирование исполняемого кода, для обработки событий.

Для формирования HTML-элементов можно воспользоваться следующим механизмом: сопоставить каждому абстрактному элементу HTML-элемент или набор HTML-элементов.

При формировании исполняемого кода стоит обратить внимание на концепцию блочного программирования, которая используется для возможности реализации функционала будущего программного продукта, а именно – на то, что каждый функциональный блок можно представить в виде лексемы или набор лексем для соответствующего языка программирования, в данном случае – JavaScript. Полученный набор лексем уже можно отправить интерпретатору или компилятору на следующий этап – этап синтаксического анализа [5,6].

Заключение

При создании инструментов визуального программирования возникают определенного рода проблемы, которые могут сделать созданный инструмент непригодным к использованию как пользователем без углубленных знаний программирования, так и самим программистом.

К сожалению, их полностью нельзя избежать, но имеется возможность снизить риск их возникновения. Для этого приходится проводить анализ смежных областей применения инструмента и оперировать, по возможности, элементарными абстракциями.

В ходе анализа проблем был предложен набор правил для минимизации возникновения проблем. На их основе предложена концепция инструмента и создан сам инструмент.

ЛИТЕРАТУРА

1. *Мартин Р.С., Нью-кирк Д.В., Косс Р.С.* Быстрая разработка программ. Принципы, примеры, практика. – М: Вильямс, 2004. – 752 с.

2. Визуальный язык ДРАКОН. Официальный сайт языка ДРАКОН, 2017. – URL: http://drakon.su (24.03.2018).

3. VisSim – система динамического моделирования / Новые информационные тех-нологии и программы, 2016. – URL: http://pro-spo.ru/winmat/480-vissim (01.04.2018).

4. VAX – инструмент для визуального программирования, или как написать SQL мышкой / Хабрахабр / Издательство интернет-проектов для IT-специалистов, 2006–2018. – URL: https://habrahabr.ru/post/333750/ (04.04.2018).

5. *Хантер Р.* Основные концепции компиляторов. – М.: Вильямс, 2002. – 256 с.

6. *Ахо А.В., Лам М.С., Сети Р.* Компиляторы: принципы, технологии и инструментарий. – М.: Вильямс, 2011. – 1184 с.

КЛЕТОЧНО-АВТОМАТНОЕ МОДЕЛИРОВАНИЕ РАСПРОСТРАНЕНИЯ ЗВУКА В ЗАДАЧЕ ПОИСКА ОПТИМАЛЬНОГО РАСПОЛОЖЕНИЯ АКУСТИЧЕСКИХ СИСТЕМ

Я.И. Шмаков, М.Н. Головчинер
Томский государственный университет
yaroslavshmakovrus@gmail.com, golovchiner@mail.ru

Введение

В процессе поиска оптимального расположения акустических систем решаются две задачи [1]:

1. «Вычисление оптимального положения громкоговорителей относительно фронтальной стены».
2. «Определение расстояния от фронтальных громкоговорителей от боковых стен».

Обе задачи решаются инженером-акустиком самостоятельно, либо согласно рекомендациям производителя акустических систем по размещению акустических систем (АС).

Существуют программные комплексы, позволяющие моделировать и визуализировать распространение звука. Проблема использования таких программ – платный функционал, невозможность расширения функционала приложения.

Поэтому было решено написать своё программное обеспечение для моделирования распространения звука.

1. Постановка задачи

Стандартным подходом к решению задачи моделирования распространения звука является проведение точных численных расчётов путей распространения звука с учётом эффектов отражения, преломления, рассеивания [2] (как реализовано, например, в