УДК 004.023

Д.В. Жевнерчук, А.С. Захаров

СЕМАНТИЧЕСКОЕ МОДЕЛИРОВАНИЕ ГЕНЕРАТОРОВ ПРОГРАММНОГО КОДА РАСПРЕДЕЛЕННЫХ АВТОМАТИЗИРОВАННЫХ СИСТЕМ

Нижегородский государственный технический университет им. Р.Е. Алексеева

Разработка автоматизированных распределенных систем стратегического, корпоративного уровня, создание на их основе прикладных сервисов является сложным стохастическим процессом, требующим, в том числе, написания повторяющегося программного кода, а также типовых программных конструкций, реализующих шаблоны программирования, характерные для той или иной задачи.

Современные средства разработки зачастую накладывают дополнительные идеологические, синтаксические, структурные ограничения. Кроме того, в ходе разработки необходимо выполнять сопряжение программного кода, написанного на разных языках программирования и разметки.

В работе предлагается компонентное представление генератора многомодульного, многоязыкового программного кода, его семантическая модель и методика применения при разработке программного обеспечения широкого класса.

Ключевые слова: семантическое моделирование, интерфейс, генерация кода, конфигурирование.

Введение

Одним из свойств теории вычислимости (теории рекурсивных функций) является полнота по Тьюрингу. Это свойство заключается в том, что для каждой вычислимой функции существует такой элемент, который может эту функцию вычислить [1]. Иными словами, приводя это определение в области генерации программного кода, получаем следующий факт: если программа может быть написана, то существует такой элемент, который способен ее написать. Далее такой элемент будем называть генератором программного кода (ГПК).

В настоящее время все более актуальной задачей становится создание программного продукта, позволяющего автоматизировать труд разработчика прикладного программного обеспечения. Под понятием разработчик в данной статье подразумевается участник команды, который принимает участие в создании программного продукта на любом этапе. При проектировании разработчики стараются сделать структуру приложения максимально простой и стандартизированной - для облегчения программирования и сопровождения. Поэтому, при старте нового проекта, разработчику зачастую приходится выполнять однотипные действия по созданию базового шаблона той или иной конструкции. При правильных входных данных и корректной разработке генератора, большая часть кода может быть сгенерирована автоматически. Разработчику же останется внести небольшие правки в итоговый код программы.

Генераторы кода лучше всего применимы в тех проблемных областях, в которых выполняются небольшие по объему и одинаковые по сути операции [2]. Потенциально генераторы могут применяться в любых областях (задачах), где можно автоматизировать процесс работы с данными. Наиболее близкими к этому описанию являются задачи генерации кода проекта, взаимодействующего с базой данных, имеющего сходные классы и структуры для объектов, приложения для тестирования программного кода или написания документации по существующей программе.

Аналитическая часть

Для того чтобы сократить время для разработки стартовой (начальной) части проекта, используются автоматизированные системы (АС) генерации программного кода. Генерация кода с помощью метода, описываемого в данной статье, осуществляется в несколько этапов:

шаблонизации, конфигурирования, рендеринга. Для понимания этапов введем следующие определения.

Определение 1. ***Шаблон*** – это структурированный текст, определяющий блок программного кода, в котором выделены неизменяемые конструкции, характерные для используемого языка программирования, а также поля, которые могут быть заменены типами и именами используемых переменных, областями видимости, начальными значениями и т.д.

Под ***шаблонизацией*** будем понимать процесс формирования системы связанных друг с другом шаблонов.

Определение 2. ***Конфигурирование*** – это процесс формирования пар, первым элементом которых является шаблон, а вторым – вектор данных, называемый также ***конфигурацией***, используемый для замены полей шаблона.

Определение 3. Рендеринг – это процесс замены полей шаблона полями конфигурации с последующим формированием готового блока программного кода.

Общая схема генерации программного кода включает в себя два этапа. *На первом этапе* происходит шаблонизация и конфигурирование. На вход шаблонизатора передаются конструкции языка и изменяемые поля, на выходе же получается шаблон. Для процесса конфигурирования входными параметрами являются модели предметной области, на выходе получается итоговая конфигурация. Модели предметной области формализуют структурные и функциональные аспекты автоматизируемой предметной области и являются источником используемых в автоматизируемой системе имен, типов, ограничений. *На втором этапе* из полученных шаблона и конфигурации происходит рендеринг итогового программного кода.

В ходе разработки автоматизированной системы, кроме ГПК, применяются еще генераторы, которые создают базовый прототип программной системы, включающий требуемую структуру папок, конфигурационные файлы, модули с описанной интерфейсной частью, различные дополнения и их настройки (модули с готовым функционалом, ресурсы, плагины). В зависимости от технологий, языков программирования и платформ существуют различные инструменты, решающие эту задачу [3].

Для создания автоматизированных систем, ориентированных на известный круг задач, применяют каркасный подход, предполагающий построение исходного прототипа на базе программной платформы (фреймворка), состоящего, как правило, из постоянной части, называемой ядром, и сменных модулей или точек расширения.

Наиболее известным инструментом для быстрого создания стартового проекта является скаффолдер Yeoman. *Скаффолдинг* – это метод метапрограммирования, при котором проект генерируется на базе анализа требований разработчика [4]. Основная функция метапрограммирования – это использование программ для создания других программ. Yeoman базируется на трех основных компонентах: менеджер зависимостей пакетов, необходимый для загрузки, обновления и удаления дополнительных пакетов (bower); инструмент для сборки java-script проектов с использованием заранее написанных задач (grunt); базовое приложение, отвечающее за генерацию базы для нового приложения (yo).

Настройка всего проекта выполняется одной командой, которая запускает один из более 5500 генераторов кода, описанных на сайте [4]. Например, для проекта angular необходимо выполнить команду**:** "yoangular", после чего генератор задает вопросы относительно компонент и генерируется проект (с правильной структурой файлов и директорий), который сразу же можно запустить на 9000 порту и открывает стартовую страницу приложения командой: gruntserver.

Для разработчиков на платформе dotNet компания Microsoft разработала генератор кода Text Template Transformation Toolkit или T4 [5] на базе шаблонов. Для использования данного генератора необходимо написать файл-шаблон в формате *.tt. Существует большое количество готовых шаблонов для генерации стартового проекта под необходимые требования, однако из-за отсутствия единого репозитория поиск этих шаблоном крайне затруднен. Кроме того, T4 используется исключительно для генерации только C# кода.

В стеке Java-технологий для генерации рутинного повторяющегося программного кода для решения таких типовых задач, как создание кода для конструкторов классов с большим количеством полей, создание и редактирования JavaDocs, геттеров/сеттеров, управление кодом коллекций и сборщиков композиционных объектов, управление кодом для тестирования также применяются многочисленные генераторы. Одни встраиваются в интегрированные среды разработки, другие являются самостоятельными системами, поставляемыми в виде, например, jar файлов: FreeBuilder, Proto. Интеграция внешних генераторов зачастую требует написания своего специфического рутинного кода, исчисляемого десятками строк.

Для того чтобы развернуть приложение на Java (Spring), необходимо подключить большое количество различных классов – bean. Кроме того, в зависимости от способа сборки проекта, необходимо найти дополнительные библиотеки и так же их подключить к существующему проекту. Для генерации стартового проекта можно воспользоваться платформой Grails [6], с помощью которой можно сгенерировать готовый проект используя только одну команду: "grailscreate-appapp-name".

В мире Web-разработки известны надстройки, называемые "синтаксический сахар", которые способствуют сокращению количества строк исходного кода за счет его генерации. К ним можно отнести SugarJS, CoffeeScript [7]. Подобные надстройки транслируют код, написанный на новых языках программирования, в код, написанный на языке Java-script. Самостоятельные шаблонизаторы, такие как Underscore templates, HandleBars, позволяют хранить параметрически настраиваемые шаблоны html страниц и их фрагментов на сервере, и по мере необходимости генерировать html код и пересылать его клиенту.

Итак, можно сделать выводы о том, что современные подходы и средства генерации кода не эффективны при разработке современных распределенных автоматизированных систем (АС), решающих задачи корпоративного, стратегического уровня, поскольку:

- трудно адаптируются для создания программного кода двух и более интероперабельных модулей, написанных с использованием множества языков программирования и форматов представления данных,
- в большинстве случаев ориентированы на выполнение низкоуровневой генерации по известным шаблонам, требуют изучения уникальных архитектур, объектных моделей, нотаций и пр. для использования и модификации генератора под новые задачи,
- как правило, не могут быть применены для генерации кода при алгоритмических и языковых ограничениях, включая сопряжение с существующим кодом,
- в общем случае внесение изменений в системы генерации кода невозможна без перекомпиляции системы.

Постановка задачи

Была поставлена задача, которая заключается в разработке компонентного представления генератора гибридного (многомодульного, многоязыкового) программного кода, а также его семантической модели. Каждый компонент генератора должен обладать интерфейсами, посредством которых сопрягается: а) с другими компонентами-генераторами, б) с конфигураторами, используемыми для рендеринга шаблонов кода. Кроме того, каждый генератор может быть использован как самостоятельно, так и в качестве звена многокомпонентных цепочек.

Теоретическая часть

Согласно принципам построения открытых информационных систем, ГПК должен обладать свойствами расширяемости, масштабируемости, кроссплатформенности и интероперабельности [8] и представлять собой совокупность компонент с выделенными интерфейсами. Введем несколько определений.

Определение 4. ***Компонент ГПК*** – абстрактный структурный элемент генератора программного кода, обладающий точками входа/выхода и связанными с ними элементар-

ными свойствами, к которым относятся изменяемые поля, определяющие интерфейсы ввода конфигурации и вывода программного кода как результата рендеринга.

Введем два типа компонентов ГПК.

Определение 5. ***Рендер программного кода (РПК)*** – структурный элемент генератора программного кода, способный принимать конфигурацию и на основе шаблона возвращать программный код.

Определение 6. ***Конфигуратор программного кода (КфПК)*** – структурный элемент генератора программного кода, способный принимать от клиентских систем конфигурацию и передавать ее РПК.

На рис. 1 представлена концептуальная схема компонентной организации ГПК распределенных автоматизированных систем.

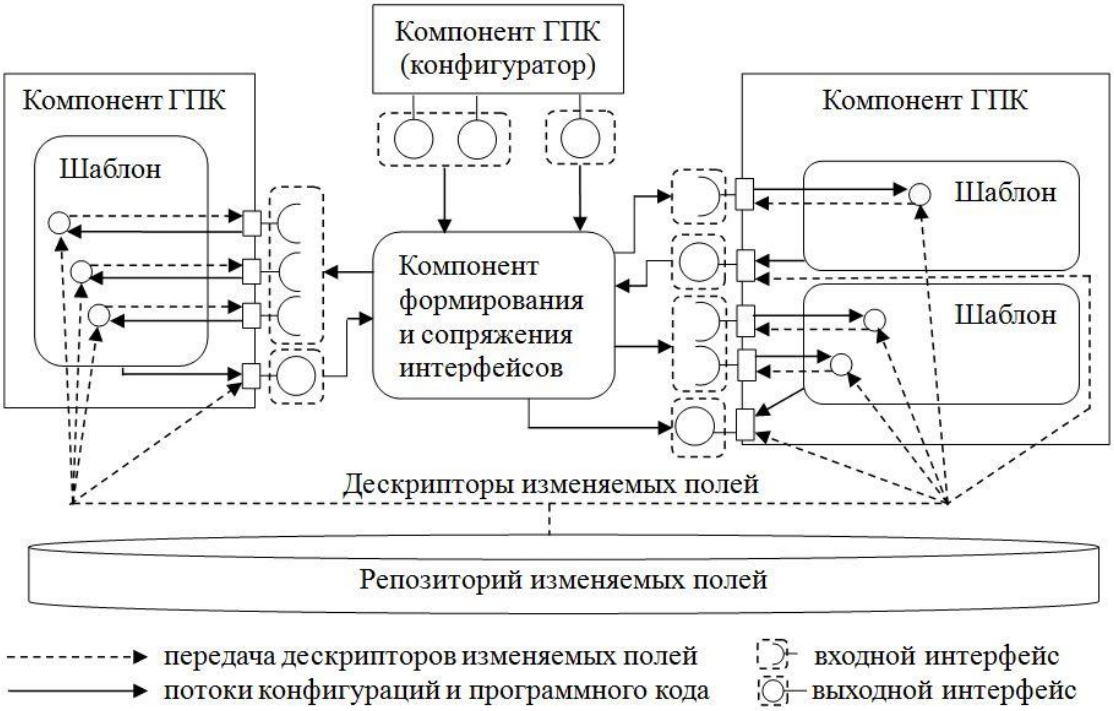

Рис. 1. Концептуальная схема ГПК АС

РПК и КфПК содержат в качестве своих элементов шаблоны. Шаблоны РПК включают неизменяемые программные конструкции и уникальные дескрипторы изменяемых полей, посредством которых возможна их идентификация. Шаблоны КфПК включают только перечень дескрипторов изменяемых полей, для которых клиентская система должна подготовить конфигурацию.

Кроме дескрипторов изменяемых полей, шаблон может обладать и другими элементарными свойствами, что позволяет строить интерфейсы, посредством которых компонент ГПК может быть сопряжен: с моделями логического и даталогического уровня, такими как алгоритмы, модели состояний, ER-модели, модели классов и др., классификаторами и блоками, определяющими семантические, технические, лицензионные и иные ограничения, блоками мониторинга режимов использования АС.

Таким образом, методика синтеза открытых информационных систем позволяет представить генератор программного кода и его внешнее окружение в виде типовых компонент, обладающих унифицированными интерфейсами, синтезируемыми на основе элементарных свойств.

Пунктирными стрелками показаны потоки дескрипторов изменяемых полей, образующиеся при создании шаблонов, а также при синтезе интерфейсов с помощью компонента формирования и сопряжения интерфейсов.

Потоки конфигураций и результатов рендеринга шаблонов, обозначенные сплошными линиями, возникают в системе при решении задачи генерации программного кода.

Пусть компонент ГПК c_r содержит шаблон, на основе которого необходимо сформировать программный код, и он определяется свойствами $T_{c_r} = (t_{c_{r_1}}, t_{c_{r_2}}, \dots, t_{c_{r_n}})$, а компоненты ГПК $C = (c_1, c_2, \dots, c_j, \dots, c_s)$ являются потенциальными поставщиками конфигураций и определяются свойствами своих шаблонов $T_{c_j} = (t_{c_{j_1}}, t_{c_{j_2}}, \dots, t_{c_{j_n}})$, для любого *j*. Рассмотрим особенности сопряжения c_r и c_j, где $j=1..s$.

Возможны три основных случая:

а) $|T_{c_r}|=0$, шаблон компонента c_r не содержит ни одного дескриптора изменяемого поля. Обладает выходным интерфейсом, связанным с дескриптором изменяемого поля, используемым шаблонами внешних компонент ГПК, через который передается во внешнюю среду неизменяемая строка с готовым программным кодом. Код шаблона c_r является готовым программным кодом;

б) $|T_{c_r}|=1$, шаблон компонента c_r содержит одно изменяемое поле, для его рендеринга достаточно одного компонента ГПК $c_o \in C$, причем должны выполняться следующие условия:

1) $\exists\, q \in T_{c_r}\ p \in T_{c_0}: Id_q = Id_p$(для компонент c_r*и* c_o существуют элементарные свойства, определяющие изменяемые поля с одинаковыми дескрипторами);

2) q является дескриптором изменяемого поля c_r, т.е.определено на множестве точек входа блока c_r, а p является дескриптором результата рендеринга, т.е. определено на множестве точек выхода c_o соответственно;

в) $|T_{C_r}| >1$, шаблон компонента c_r содержит более одного дескриптора изменяемого поля, для его рендеринга необходимо несколько компонентов ГПК $C_o \subseteq C_j$, причем должны выполняться следующие условия:

1) $\forall q \in T_{c_r}\ \exists p \in T_{C_0}: Id_q = Id_p$(для любого изменяемого поля шаблона компонента c_r существует элементарное свойство компонента из множества C_o, определяющее изменяемое поле с таким же дескриптором);

2) все элементарные свойства q являются дескрипторами изменяемых полей c_r, т.е. они определены на множестве точек входа блока c_r, а все p являются дескриптором результата рендеринга, т.е. определены на множестве точек выхода компонент из множества C_o соответственно.

На основании концептуальной схемы предложен набор концептов и ролей, с помощью которого может быть построена семантическая модель ГПК распределенных автоматизированных систем.

Методическая часть

Согласно предложенной концептуальной схемы генерации программного кода, была построена семантическая модель ГПК, фрагмент которой представлен на рис. 2, определяющая пространство имен концептов и ролей. Модель содержит базовые концепты, формализующие рендер и конфигуратор программного кода, как подклассы компонента ГПК, с которым связаны точки входа/выхода потоков данных.

С каждой точкой может быть связано некоторое множество элементарных свойств, определяющих изменяемые поля и характеризующиеся дескриптором, базовым типом и ограничениями. Следует отметить, что дескриптор и базовый тип не связаны с другими концептами и поэтому могут быть реализованы простыми атрибутами, а ограничение реализуется классом, имеющим подклассы, с помощью которых можно указать допустимые, запрещенные списки и диапазоны значений.

Кроме того, рендер и конфигуратор содержат шаблоны, реализуемые простыми атрибутами и представляющими собой строки, содержащие код программных конструкций и параметры-подстановки, имеющие значения дескрипторов соответствующих изменяемых полей. Изменяемые поля могут быть сгруппированы с помощью концепта, задающего интерфейса.

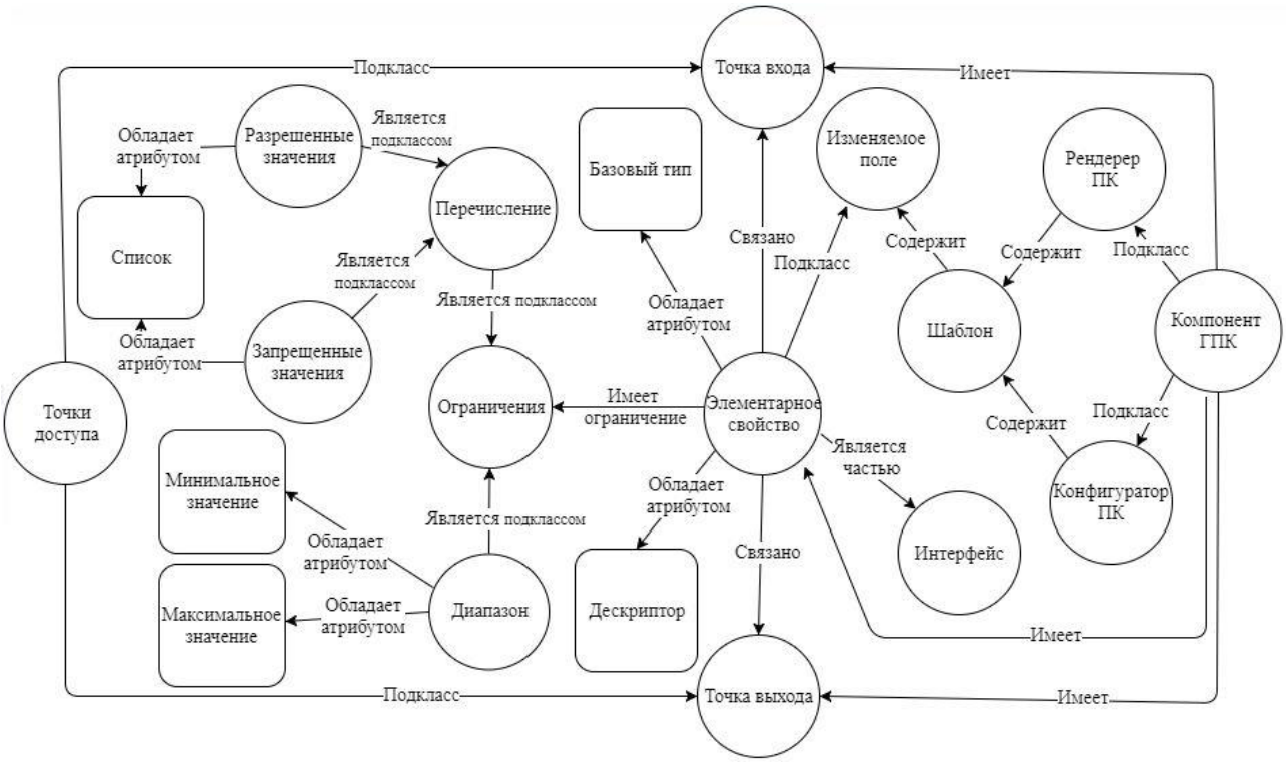

Рис. 2. Фрагмент семантической модели генератора программного кода

На рис. 3 представлено схематическое описание методики синтеза программного кода.

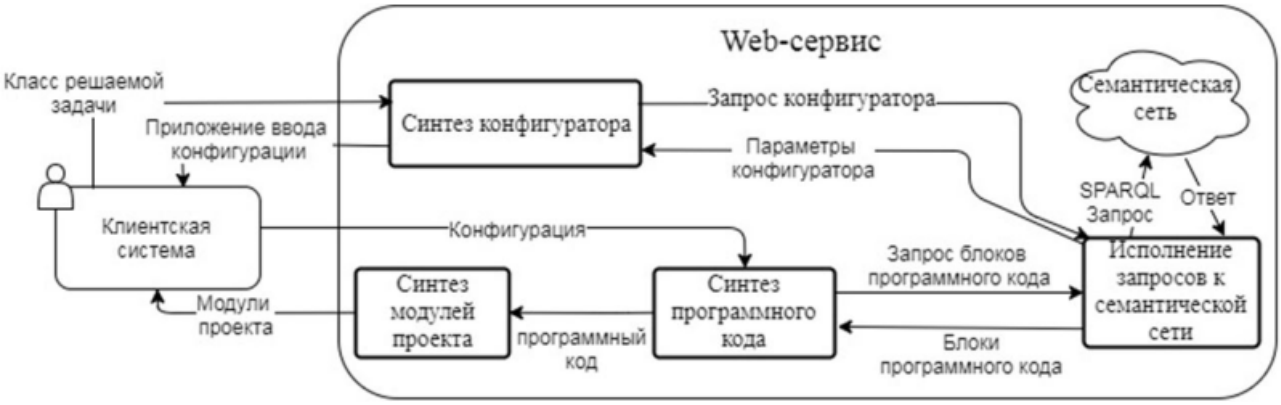

Рис. 3. Схема описания методики синтеза исходного кода проекта

На первом шаге формируется запрос, определяющий класс решаемой задачи и содержащий, кроме того, требования, предъявляемые к проекту, язык программирования, особенности архитектуры, особенности самого приложения (с точки зрения безопасности или приложений реального времени). На втором шаге по полученным данным создаются запросы к семантической сети, которые возвращают параметры конфигурации и на их основе формируется клиентское приложение для ввода конфигурации.

На третьем шаге клиентская система передает конфигурацию, на основе которой выполняется синтез программного кода. В ходе синтеза формируется последовательность запросов к семантической сети, возвращающих цепочку связанных РПК и КфПК, определяющую параметры ГПК, используемые для настройки объектов синтеза кода. На последнем шаге готовый код размещается в файлах-модулях целевого проекта автоматизированной системы, которые определенным образом структурируются, объединяются в пакеты, располагаются в соответствующих папках. Правила группирования зависят от используемых техноло-

гий программирования, фреймворков, принятых архитектурных решений. Готовые модули передаются клиенту.

Экспериментальная часть

Для подтверждения теоретических и методических аспектов работы была проведена серия экспериментов с прототипом ГПК. В частности, были построены модели конкретных генераторов программного кода: POJO, JPA Entity, HTML form, backbone js model/collection. На рис 4 представлен фрагмент семантической сети ГПК POJO.

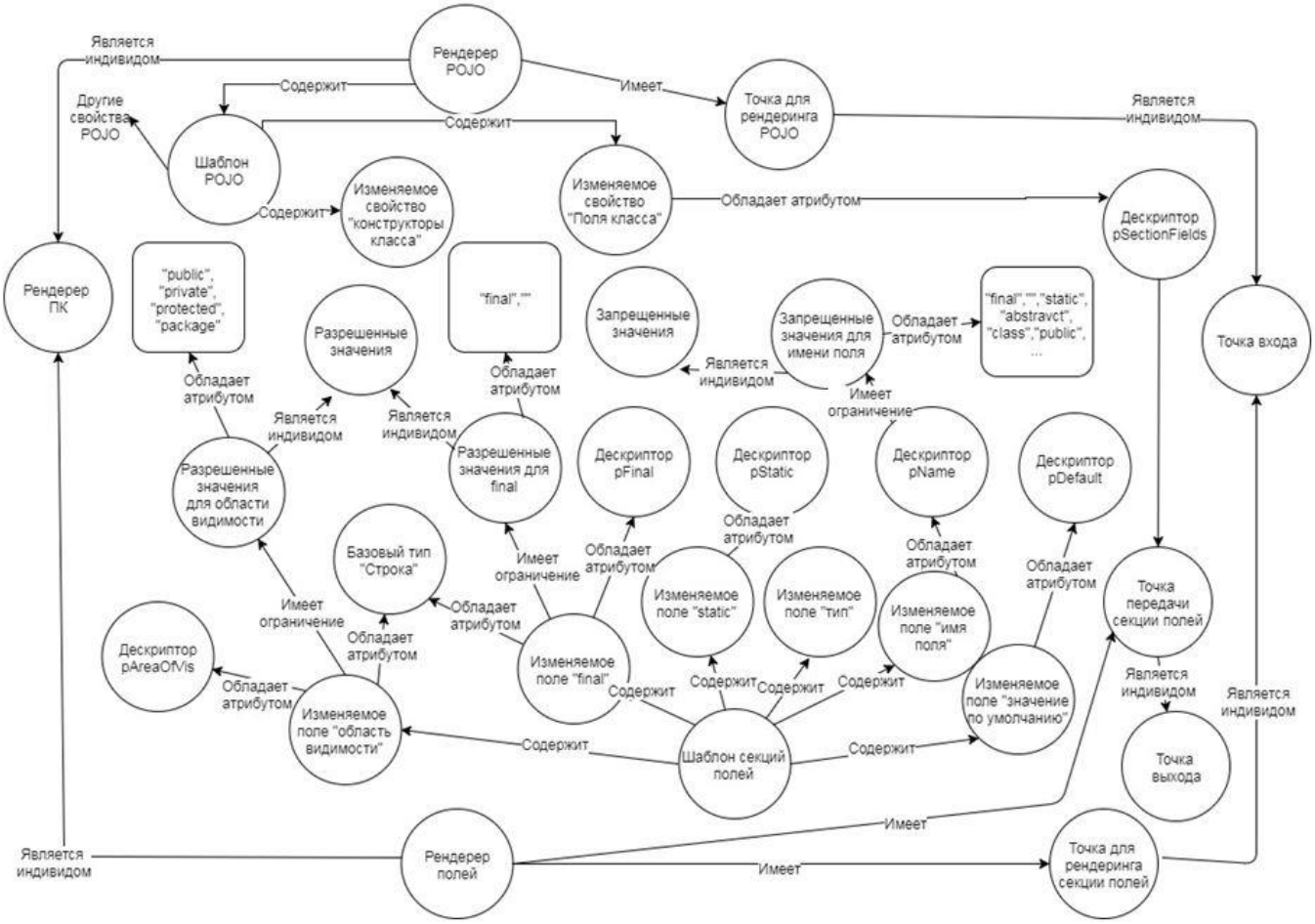

Рис. 4. Фрагмент семантической сети ГПК POJO

На основании построенных семантических моделей был сгенерирован программный код, успешно прошедший компиляцию и серию тестов.

Тем не менее, в процессе построения семантических моделей процент ошибок, связанный с человеческим фактором, составил 29%. Это в первую очередь связано с большим объемом вводимых данных, отсутствием алгоритмов контроля коллизий, возникающих в ходе ввода и редактирования данных, а также при выполнении правил логического вывода.

Выводы

В ходе работы была проведена адаптация методики синтеза открытых информационных систем к генераторам программного кода, что позволило обеспечить их свойствами масштабируемости, интероперабельности и кроссплатформенности. Построены и экспериментально проверены семантические модели ГПК, которые, в дальнейшем, могут быть реализованы в виде баз знаний онтологического типа, что позволит описывать сценарии генера-

ции программного кода, независимые от используемого комплекса языковых средств и технологий шаблонизации.

Библиографический список

1. **Евгенев, Г. Б.** Интеллектуальные системы проектирования: учеб. пособие / Г. Б. Евгенев. − М.: Изд-во МГТУ им. Н.Э. Баумана, 2009. – 334 с.
2. **Кустов, М.Н.** Повышение эффективности алгоритмов генерации программного кода по методологии SADT. Информационные интеллектуальные системы // Радиоэлектроника и молодежь в XXI веке: материалы XVI международного молодежного форума. – Харьков: ХНУРЭ, 2012. – Т. 6. – С. 228–229.
3. **Клыков, Ю. И.** Ситуационное управление большими системами / Ю. И. Клыков. – М.: Энергия, 1974. – 134 с.
4. The Web's scaffolding for modern webapps [Электронный ресурс] // 2017. − Режим доступа: http://yeoman.io/ (дата обращения 31.07.2017)
5. **Ross, D.** Structured Analysis (SA): A Language for Communicating Ideas, IEEE Transactions on Software Engineering // IEEE Transactions on Software Engineering. – 1977. – V. SE-3. – № 1. – P. 16–34.
6. **Basu, A.** Metagraphs and Their Applications / A. Basu, R. Blanning// Springer Computer Society Press. – 2007. – 174 p.
7. **Kelly, S.** Domain-Specific Modeling: enabling full code generation / S. Kelly, J. P. Tolvanen // Hoboken. **–** New Jersey, USA: Wiley-IEEE Computer Society Press. − 2008. − 444 p.
8. **Фаулер, М.** Шаблоны корпоративных приложений / М. Фаулер, Д. Райс. – М.: Вильямс, 2016. – 544 с.

Дата поступления
в редакцию 15.01.2018

D.V. Zhevnerchuk, A.S. Zakharov

SEMANTIC MODELING OF THE PROGRAM CODE GENERATORS FOR DISTRIBUTED AUTOMATED SYSTEMS

Nizhny Novgorod state technical university n.a. R.E. Alekseev

Purpose: Development of a component representation of a multimodule code generator. The article presents a semantic model and a technique for application in the development of software.
Design/methodology/approach: Based on synthesis of open information systems a generator needs to comply with the following requirements: extensibility, scalability, cross-platform and interoperability. A fragment of concept scheme of distributed automated systems generator is presented.
Findings: In the course of the work, the methodology of the open information systems synthesis was adapted to a program code generator, which allowed us to create scalable, interoperable and cross-platform generators.

Key words: semantic modeling, interface, code generation, configuration.