DOI dx.doi.org/10.24866/VVSU/2073-3984/2019-4/193-206

УДК 519.685, 004.656, 004.434

Д.И. Харитонов[1]

Д.С. Одякова[2]

Д.В. Леонтьев[3]

Р.В. Парахин[4]

Институт автоматики и процессов управления Дальневосточного отделения Российской академии наук
Владивосток. Россия

Генерация исходных кодов для тематических коллекций научных данных*

В настоящей статье рассматривается подход к построению электронных коллекций научных данных, основанный на автоматической генерации исходного кода для компонентов информационной системы на базе ER-модели предметной области, проектируемой дизайнером информационной системы. Подход опирается на архитектурную схему генерации, содержащую аппарат моделирования, состоящий из шаблонного графа метаязыка и прототипов элементов описаний на метаязыке; модель предметной области, состоящей из ER-модели в графическом виде и её объектного представления; жизненный цикл, состоящий из модели в терминах сетей Петри и объектного представления. В статье дано описание жизненного цикла электронной коллекции и взаимодействие компонентов информационной системы, соответствующее этому жизненному циклу, приводится описание метаязыка в виде графа, в котором определено остовное дерево и для каждого узла дерева определены роль, значение и приоритет. Для описания объектов предметной области и их взаимодействия разработан и описан в виде БНФ грамматики язык шаблонов исходного кода. На примере коллекции фораминиферов рассмотрен процесс генерации исходных кодов на целевом языке программирования. Приведены примеры фрагментов сгенерированных файлов на языке SQL для некоторых событий жизненного цикла. Рас-

[1] Харитонов Дмитрий Иванович – канд. техн. наук, ст. науч. сотрудник; e-mail: demiurg@dvo.ru

[2] Одякова Дарья Сергеевна – старший инженер-программист; e-mail: darlene@dvo.ru

[3] Леонтьев Денис Васильевич – научный сотрудник; e-mail: devozh@dvo.ru

[4] Парахин Роман Валерьевич – инженер-программист; e-mail: fadak@dvo.ru

* *Работа выполнена при поддержке программы «Приоритетные научные исследования в интересах комплексного развития Дальневосточного отделения РАН», проект 18-5-104, в рамках госбюджетной темы научных исследований № АААА-А17-117040450019-8.*

крытый в статье подход к автоматизации построения информационных систем позволяет формализовать представление пользователей о предметной области, описать интерфейс и специфицировать методы обработки данных. Разбиение процесса генерации на две стадии увеличивает степень повторного использования программного кода, а промежуточное внутреннее представление позволяет оценить и проанализировать полученный код на корректность.

Ключевые слова и словосочетания: коллекции научных данных, сети Петри, БНФ грамматики, автоматическая генерация исходного кода, ER-моделирование.

D.I. Kharitonov

D.S. Odyakova

D.V. Leontiev

R.V. Parakhin

IACP FEBRAS
Vladivostok. Russia

Source code generation for scientific data collections

The article discusses an approach to building electronic collections of scientific data, based on the automatic source code generation for information system components based on the ER-model of subject area prepared by the information system designer. The approach relies on the architectural generation scheme, containing: a modeling apparatus consisting of a metalanguage template graph and prototypes of description elements in the metalanguage; a domain model consisting of an ER-model in graphic form and its object representation; life cycle consisting of a model in terms of Petri nets and its object representation. The article describes the life cycle of the electronic collection and the interaction of the information system components corresponding to this life cycle, defines the metalanguage in the graph form in which the spanning tree is defined with the role, value and priority assigned for each node of the tree. To describe the domain model objects and their interaction, a source code template language was developed and described in terms of BNF grammar. The process of source code generation for the target programming language is considered on the example of a foraminifera collection. Examples of generated SQL files fragments for some life cycle events are given. The approach to automation of building information systems described in the article allows us to formalize the user's view of the subject area, to describe the interface and to specify data processing methods. Dividing the generation process into two stages increases the program code reusability, and the intermediate internal representation allows us to evaluate and analyze the resulting code for correctness.

Keywords: scientific data collections, Petri nets, BNF grammar, automatic source code generation, ER-modeling.

Автоматическая генерация текстов – это современный уровень развития информационных технологий, применяется в самых разнообразных условиях, начиная от автоматов, отвечающих на письма, и ботов, общающихся в чате, далее в системах автоматической генерации текстов [3], в том числе экзаменационных заданий [2], и заканчивая генераторами тестов программного обеспечения [1] и

самим программным обеспечением [13; 7]. Причины для применения генераторов разнообразны. Так, например, традиционные системы массового производства непросто перенастроить для новых продуктов вручную, и при наличии большего количества средств связи и тенденций к оцифровке окружающего мира и стремлении потребителей к более индивидуализированным продуктам желательно ускорить этот процесс. Ручное программирование роботов в режиме он-лайн может потребовать много времени на настройку, но в то же время является глобальным и дорогостоящим. Время для ручного программирования робота для некоторых процессов в несколько сотен раз может превышать время выполнения той же программы. Поэтому автоматическая генерация кода для промышленных роботов выглядит естественным направлением развития в этой индустрии.

Рассмотрим данный вопрос с другой стороны. При разработке программного обеспечения формальная спецификация требуемого поведения системы является крайне полезным инструментом. Во-первых, спецификация содержит существенно меньше деталей, чем конечная программа, поэтому для разработчика программного обеспечения спецификация легче для понимания, чем программный код, и, следовательно, в ней легче обнаружить и исправить ошибки. Во-вторых, формальная спецификация может быть проверена на корректность, в результате которой определяется соответствие спецификации критическим свойствам, а поведение конечной программы может быть промоделировано, чтобы доказать отсутствие сбоев и аварийных остановок. Однако сама по себе формальная спецификация не гарантирует отсутствие ошибок в коде программы. Так, программный код, создаваемый программистом, подвержен различного рода ошибкам из-за отсутствия непосредственной связи со спецификацией, а также вследствие человеческого фактора. Некоторая уверенность в правильности программного кода может быть достигнута с помощью тестирования, но эта уверенность в целом определяется качеством используемых тестов [14]. Логичным вариантом повышения уровня воплощения спецификации в программный является автоматическая генерация [16] исходного текста программы из спецификации, устраняющая ошибки, возникающие при ручном кодировании. Такой подход является перспективной и растущей тенденцией в современной практике программного обеспечения. Можно отметить примеры применения генераторов программ при разработке специального программного обеспечения, например, для систем реального времени [6], роботизированных систем [8], генетических алгоритмов [12], генерации тестов [1], в частности для серверного программного обеспечения и для микропроцессоров [4], а также для генерации программных фрагментов в соответствии с техническими требованиями [7]. В настоящей статье рассматривается задача генерации программного кода для создания информационных систем, накапливающих и обрабатывающих данные научных наблюдений [5]. Специфика этой задачи, во-первых, связана с распределённым характером программного обеспечения, состоящего из базы данных, методов обработки, выполняемых в фоновом режиме, веб-сервера для взаимодействия с клиентской частью, исполняемой на компьютере пользователя в интернет-

браузере. В результате появляется необходимость в достаточно универсальном методе генерации кода одновременно на нескольких языках программирования, таких как SQL, pgSQL, Python, html, javascript и т.д. Во-вторых, к специфике задачи можно отнести достаточно простое наполнение базы данных, состоящее фактически из трёх частей. Это некоторый каталог или типизатор данных, собственно наблюдения или результаты экспериментов и плюс результаты постобработки экспериментов в виде таблиц с некоторой статистикой, вычисленной на основе наблюдений. Благодаря относительно простой структуре базы данных эта задача позволяет сконцентрироваться на логике генерации кода и взаимосвязях между результирующими фрагментами кода.

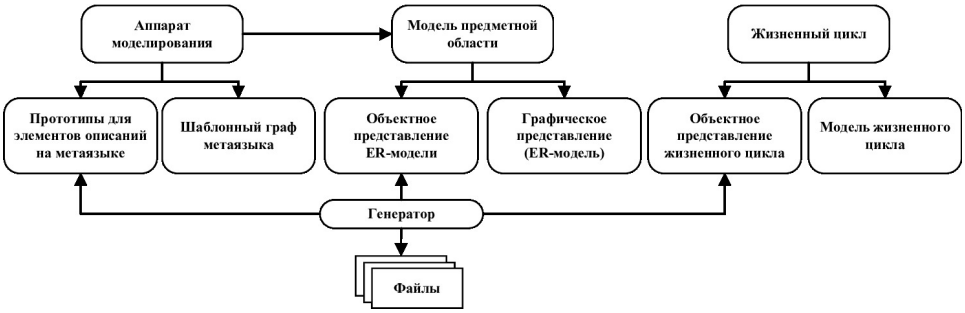

Рис. 1. Архитектурная схема генерации исходного кода

Предлагаемый в настоящей статье подход к построению электронных коллекций научных данных базируется на автоматической генерации исходного кода для компонентов информационной системы на основе ER-модели [15] предметной области, проектируемой дизайнером информационной системы. Сформированная модель предметной области передаётся подсистеме генерации программного кода, которая использует ассоциированные с элементами описания ER-модели шаблоны элементов исходного кода и алгоритмы преобразования этих шаблонов, написанные на внутреннем языке, для построения результирующего текста на целевых языках программирования.

На рисунке 1 представлена архитектурная схема генерации исходного кода, состоящая из трёх описательных разделов, используемых генератором: аппарата моделирования, модели предметной области и жизненного цикла. К аппарату моделирования относится шаблонный граф метаязыка для описания предметной области и прототипы элементов описаний (классы) на внутреннем языке, соответствующие узлам графа. Модель предметной области строится с использованием понятий, определённых в метаязыке, она состоит из графического представления ER-модели, визуализирующей взаимосвязи между элементами описания, и объектного преставления, специфицирующего свойства и поведение узлов ER-модели на внутреннем языке. Объектное представление задаётся с использованием прототипов, определённых в аппарате моделирования. Третьим описательным разделом является жизненный цикл системы, определяющий множество состояний, в которых может находиться электронная коллекция и

множество событий работы с данными коллекции и перехода между состояниями системы. Для визуального представления жизненного цикла используется аппарат сетей Петри. Событиям жизненного цикла соответствуют переходы в сети Петри, а также одноимённые объекты на внутреннем языке, используемые генератором для определения множества результирующих файлов.

Жизненный цикл электронной коллекции научных данных. Жизненный цикл является основным связующим звеном между результирующими файлами с исходным кодом. В жизненном цикле определяются события в распределённой информационной системе и реакция на эти события, которая может быть описана в различных файлах на различных языках программирования. Так, для отображения наблюдений необходимо сформировать запрос к базе данных на языке SQL, код для WEB-сервера, например, на языке Python и форму для размещения данных на языке HTML.

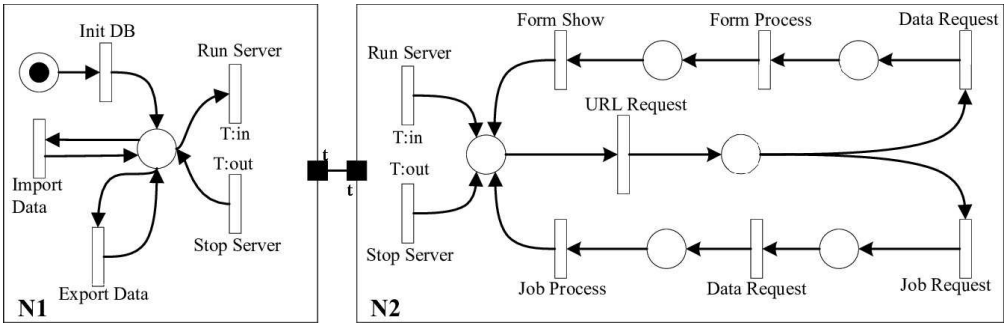

Рис. 2. Жизненный цикл электронной коллекции научных данных

Рассмотрим жизненный цикл электронной коллекции, который описан с использованием аппарата сетей Петри [10] (рис. 2). Его можно разбить на две логических части: первая отвечает за первоначальную загрузку и выгрузку данных (сеть *N1* на рисунке) и выполняется в при помощи команд в терминале, вторая (сеть *N2* на рисунке) – за интерфейс доступа и редактирования данных, как правило, эта часть осуществляется пользователем интерактивно. Сети Петри для каждой из частей изображены прямоугольниками, на границе которых размещаются точки доступа по переходам в виде закрашенного квадрата, помечены символом t (наименования точки доступа). Часть цикла, моделируемая сетью (*N1*), начинает работу с перехода *Init DB*, который отвечает за формирование структуры таблиц базы данных. Далее система переходит в «офлайн» состояние, из которого можно импортировать или экспортировать данные (переходы *Import Data* и *Export Data*). Из состояния «офлайн» может также сработать переход *Run Server*, в этом случае происходит переход по точке доступа в состояние «онлайн», которое моделируется сетью *N2*. Из состояния «онлайн» по точке доступа *t* система может обратно вернуться в состояние «офлайн».

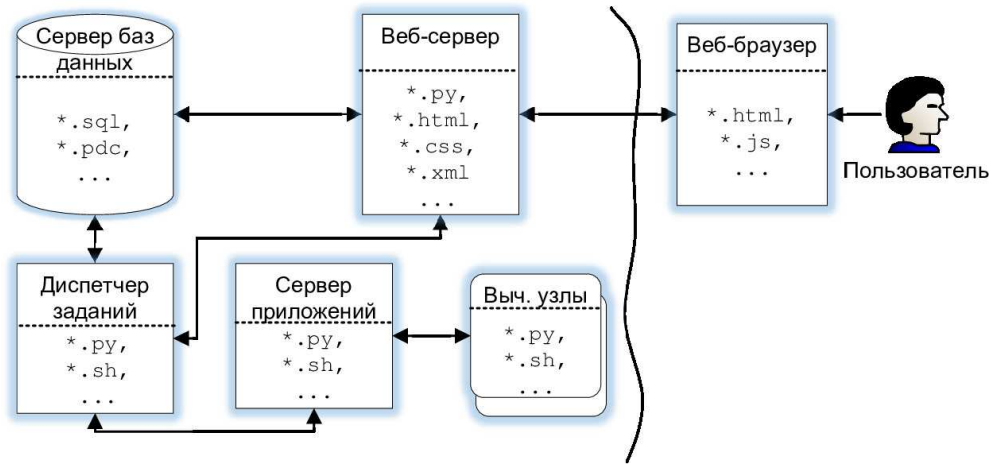

Рис. 3. Взаимодействие программных компонентов в электронной коллекции

Согласно жизненному циклу на рис. 3 представлена схема взаимодействия программных компонентов системы управления электронными коллекциями научных данных. В данной схеме выделяются две части: серверная, состоящая из веб-сервера, сервера баз данных, диспетчера заданий, сервера приложений и вычислительных узлов, и пользовательская, к которой относятся веб-браузер и пользователь. На стороне каждой из компонентов выполняется программный код на соответствующем языке программирования, который хранится в файлах, сформированных генератором. Расширение файлов для каждого из компонентов изображено на рисунке. Веб-сервер выступает связующим звеном между пользователями и остальной системой, он получает данные от сервера баз данных и диспетчера заданий, подготавливает веб-страницы, подставляет стили страниц для их отображения в веб-браузере. Задания от пользователя через веб-сервер отправляются диспетчеру заданий, который выступает в роли посредника при выборе конечного обработчика данных. Различные серверы приложений обращаются к диспетчеру заданий с сообщением о выполнении очередного задания или с запросом на следующее. Сервер приложений при необходимости может выполнить задание на многопроцессорной вычислительной системе.

Метаязык для описания предметной области. Отправной точкой построения системы автоматической генерации кода для тематических коллекций научных данных является понятие метаязыка, задачей которого является формирование инструментов описания модели предметной области, на основании которого возможна вся дальнейшая работа по автоматизации процесса генерации кода. Описание метаязыка может быть представлено в виде графа, в котором определено остовное дерево, дополненное «избыточными» связями, причём для каждого узла дерева определены роль, значение и приоритет. Приоритет определяет порядок обхода остовного дерева. Значение узла используется в качестве наименования определяемого метаязыком понятия. Роль разделяет узлы на обязательные и опциональные, узлы-списки, узлы выбора и узлы-ссылки. Таким

образом, метаязык проектируется в виде древовидного описания из максимально простых элементов, среди которых узлы-ссылки необходимы для упрощения визуального представления и сохранения древовидности структуры. Замена узлов-ссылок на связи с соответствующими вершинами дерева трансформирует остовное дерево в шаблонный граф метаязыка.

На рисунке 4 изображён шаблонный граф, предназначенный для построения ER-моделей электронных коллекций научных данных. Шаблонный граф имеет следующие основные вершины, необходимые для автоматического построения информационной системы:

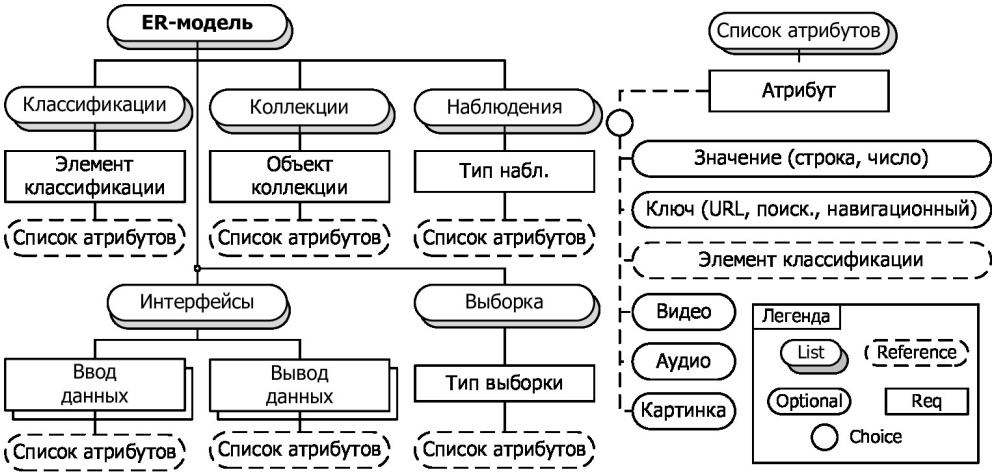

Рис. 4. Шаблонный граф метаязыка описания коллекции научных данных

Коллекция и объект коллекции – основные элементы, вокруг которых строится информационная система.

Атрибуты – минимальный элемент, с которым может работать информационная система.

Классификация и элемент классификации – единственная задаваемая уже на момент формирования информационной системы.

Наблюдения – тип вершины, предназначенный для описания собираемых данных.

Интерфейсы – предназначены для описания механизмов преобразования данных на этапах ввода информации в базу данных и на этапе формирования.

Выборка описывает варианты постобработки данных наблюдений, имеющие самостоятельную ценность.

Язык шаблонов исходного кода. Генератор формирует исходный код для каждого объекта ER-модели предметной области. Для этого выполняется обход вершин графа описываемого объекта, во время которого посещение вершины выполняет операцию преобразования шаблонного кода к результирующему подстановкой параметров и выполнением внутренних команд шаблонного языка.

Для описания шаблонов исходного кода используется xml-подобный внутренний язык. В первом приближении каждая сущность этого языка представляет собой либо список, либо параметр со значением. На рисунке 5 синтаксис языка шаблонов исходного кода представлен в форме Бэкуса-Наура [9]. Список обозначен *TreeNode*, у него есть имя, в дополнении к которому могут быть указаны тип и в круглых скобках значение. В фигурных скобках перечисляются параметры или другой список, обозначенный продукцией *Content*. Параметры (*LeafNode* на рис. 5) представляют собой двойку или тройку, состоящую из наименования и значения, в качестве третьего необязательного параметра можно указать тип. Разбор шаблона согласно грамматике, представленной на рис. 5, гарантирует структурную целостность описания и выполняется на первом этапе компиляции шаблона.

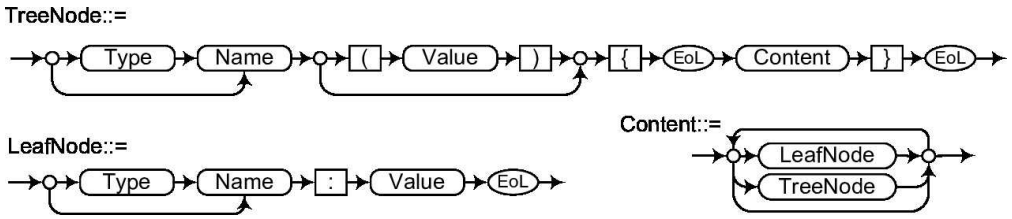

Рис. 5. Детализация элементов в BNF грамматике шаблонного языка

На втором этапе компиляции шаблона выполняется поиск семантически значимых выражений. Для этого этапа разработан язык, БНФ грамматика которого изображена на рис. 6. Описания шаблонов на этом языке описываются продукцией *Tlang*, разбивающей описание на конструкции трёх основных типов. Во-первых, это *TextLine*, которая является единственной конструкцией, явно содержащей текст шаблона исходного кода, возможно с подстановками параметров. Для подстановки параметров используется продукция *Substitution*, описывающая либо глобальную, либо локальную подстановку, в последнем случае имя параметра экранируется символом *$*. Для поиска локальных параметров используется контекст, с которым выполняется преобразование шаблона генератором. Для поиска глобальных параметров используется описание предметной области. Во-вторых, *CmdLine* – это строка, содержащая команду генератору. В настоящий момент реализована только команда *INCLUDE*, позволяющая добавить в этом месте исходный текст, сгенерированный по шаблону, который может быть указан с локальным или глобальным именем и использовать локальный или глобальный параметр в качестве контекста.

Третьей конструкцией является *CmdList*, которая используется для выполнения композициональной трансформации шаблонного текста, обозначенного в продукции нетерминальным символом *Tlang*. В настоящий момент реализованы варианты условной и циклической трансформации. За условную трансформацию отвечает продукция *CmdIf*, которая в зависимости от значения *Expression* разрешает или запрещает вставку соответствующего шаблонного текста. За циклическую трансформацию отвечает продукция *CmdForAll*, которая выполняется

либо для всех элементов контекста шаблона, либо для всех элементов объекта, определяемого нетерминалом *Substitution*, в опциональной ветке продукции *INSIDE*. Нетерминал *Record* определяет имя локального объекта в контексте трансформации шаблонного текста. Опциональным параметром *CmdForAll* является *Delim*, определяющий разделитель, который необходимо вставить между шаблонными текстами различных объектов *Record*.

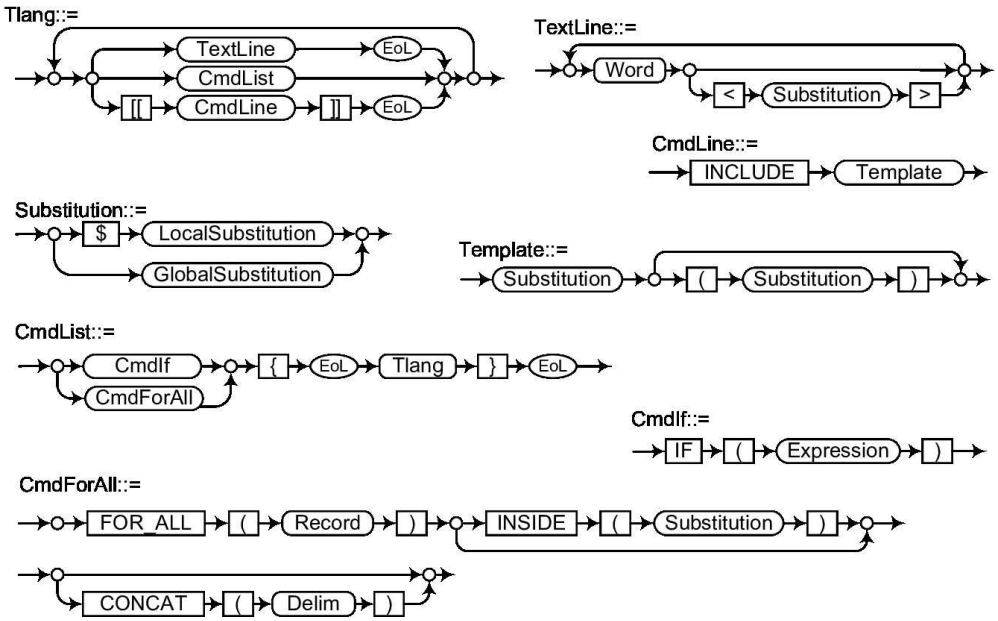

Рис. 6. Схематичное представление BNF грамматики шаблонного языка

Пример генерации кода для тематической коллекции фораминиферов. На примере тематической коллекции фораминиферов рассмотрим шаблоны исходного кода для генерации файлов, необходимых для этапа «офлайн» жизненного цикла. Для генерации кода электронной коллекции необходимо сформировать объект типа *SubjectArea*, включающий в себя *LifeCycle* – объектное представление жизненного цикла, *DataModel* – объектное представление ER-модели и *iModel* – внутреннее представление предметной области. Внутреннее представление разделяется на множества объектов, отличающихся по функциональному предназначению. Генерация исходного кода выполняется скриптом *GenerateCode* объекта *SubjectArea* в две стадии. На первой стадии каждый объект *DataModel* формирует собственное представление в каждом из множеств объектов различного функционального предназначения. На второй стадии формируются выходные файлы, соответствующие событиям жизненного цикла. Идеологически к моменту формирования файлов все необходимые для генерации данные должны присутствовать во внутреннем представлении предметной области.

На рисунке 7 представлены прототипы для описания шаблонов исходного кода на внутреннем языке. На правой части рис. 7 изображены прототипы объ-

ектов, относящиеся к дереву с корневой вершиной *Список атрибутов* на рис. 4. Базовым объектом для всех прототипов является *BaseField*, остальные прототипы атрибутов являются его наследниками. При описании атрибутов в ER-модели *BaseField* всегда будет относиться к прототипам объектов атрибутов. Так как в настоящей статье рассматривается только этап «офлайн» жизненного цикла, связанный с работой с SQL-сервером, то в базовом прототипе определяются два параметра, необходимые для создания и обращения к таблицам на сервере. Параметр *SQL_Type* определяет тип данных для поля таблицы, в котором будет храниться значение атрибута, а параметр *FieldName* – имя этого поля. Во внутреннем языке предусмотрен скрипт *Autorun*, который автоматически выполняется в момент заведения объекта атрибута. В базовом прототипе этот скрипт копирует имя атрибута в имя поля таблицы. На левой части рисунка изображены прототипы объектов, относящиеся к дереву с корневой вершиной ER-модель на рис. 4. Например, вершине с наименованием *Классификация* соответствует запись *ErmObject ErmClassification*. У прототипа *ErmClassification* есть следующие внутренние параметры: *Desc* – для описания конкретной классификации; *Instance* – для описания элемента классификации; *ArchList InitialValues* – для начального наполнения базы данных. Вершине *Интерфейсы* – *ErmObject ErmInterface*, у этой вершины есть наследные вершины, каждая из которых является отдельным объектом (*ErmInterface ErmInputInterface* и *ErmInterface ErmOutputInterface*).

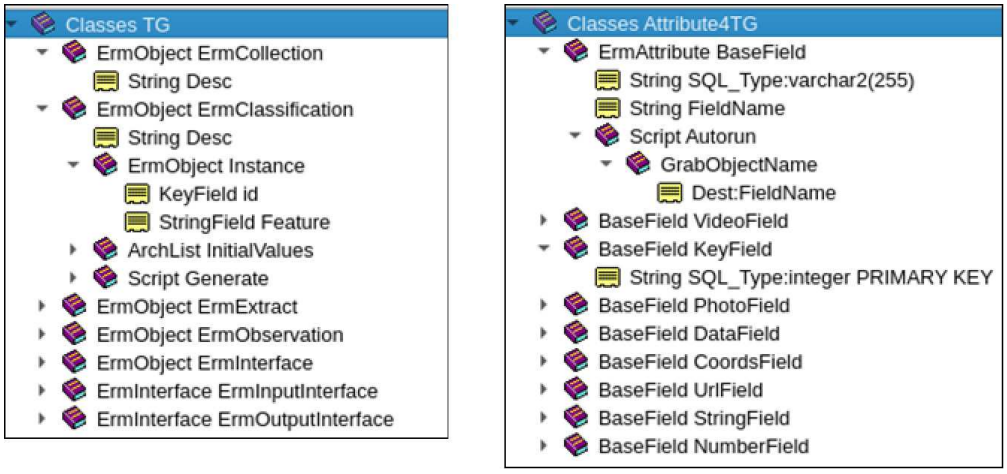

Рис. 7. Прототипы для описания на внутреннем языке

На левой части рис. 8 изображено объектное представление ER-модели. К нему относятся классификации *Тип скелета* – *SceletonType*, *Солёность среды* – *EnvSaltiness*, *Среда обитания* – Habitat, *Раковины* – ShellType. Основной объект коллекции *Foraminifera*. Наблюдениям *Контроль* и *География* соответствуют объекты *Probe* и *Geo*. Входные интерфейсы *GeoIn* и *ProbeIn* для импорта данных наблюдений в информационную систему, а также выходной интерфейс

MapOut, предназначенный для формирования карты наблюдений. На правой части рисунка изображено объектное представление жизненного цикла. Каждому объекту типа *LcEvent* соответствует одноименный переход сети *N1* на рис. 2. *LcEvent*-объекты содержат список шаблонов генерации файлов исходного кода – *SourceCodeFile*. Первым параметром шаблона генерации исходного кода является наименование множества объектов во внутреннем представлении предметной области с опциональным указанием имени обрабатываемого объекта. Вторым параметром является маска имени результирующего файла, которая может включать в себя значение любой переменной обрабатываемого объекта. Во время генерации кода, соответствующего событию, описываемому переходом *LcEvent*-объекта, у объекта вызывается скрипт *Generate*. Этот скрипт, во-первых, для каждого внутреннего объекта типа *SourceCodeFile* находит соответствующее множество объектов во внутреннем представлении предметной области; во-вторых, для каждого элемента множества создает или открывает на дополнение файл и вызывает процедуру генерации кода.

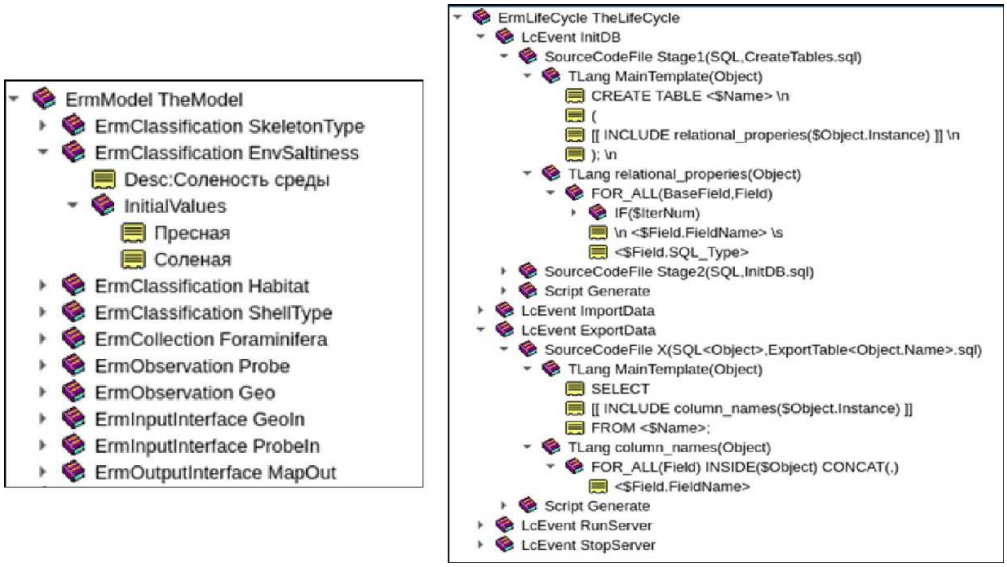

Рис. 8. Объектное представление ER-модели и жизненного цикла

Метод *GenerateCode* объекта *SubjectArea* позволяет на основании объектного представления ER-модели и жизненного цикла на рис. 8 получить файлы на целевом языке программирования. На рисунке 9 изображены примеры генерируемых файлов на языке SQL. Переходу *InitDB* жизненного цикла соответствуют два файла *CreateTable.sql* и *InitDB.sql*, шаблоны которых описаны в *Stage1* и *Stage2 LcEvent*-объекта. На правой части рисунка изображены файлы, соответствующие переходу *Export Data* и предназначенные для выгрузки данных из таблиц сервера баз данных. Пример на рис. 9 показывает, что при генерации файлов по событию жизненного цикла *InitDB* формируются два файла, каждый из которых наполняется кодом для множества объектов, тогда как по событию *Export*

Data генерируется множество файлов, каждый из которых соответствует отдельному объекту.

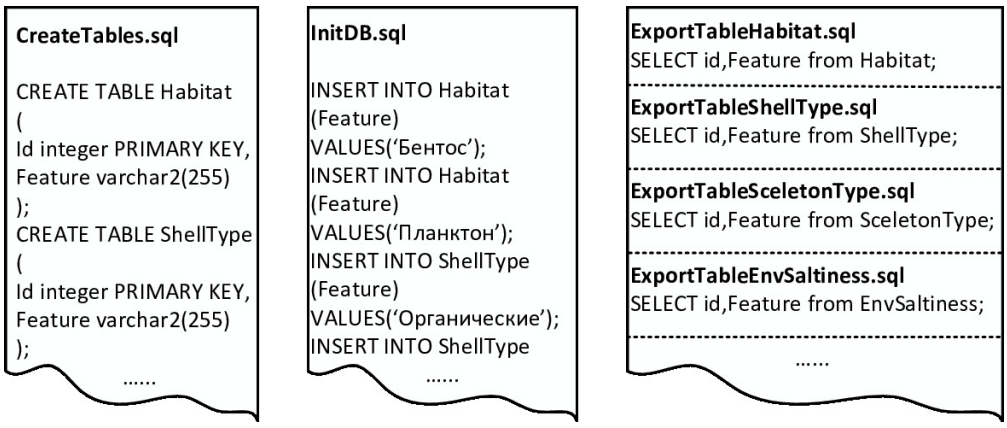

Рис. 9. Пример исходных кодов программ на языке sql

Заключение. В настоящей статье представлен метод автоматической генерации исходного кода программных элементов распределённой информационной системы, ориентированной на коллекции научных данных. Исходными данными для метода служит описание предметной области в виде объектного представления ER-модели и жизненного цикла. ER-модель предназначена для определения множества сущностей в электронной коллекции, взаимосвязей между различными сущностями, способов обработки и отображения сущностей. Жизненный цикл определяет множество состояний, в которых может находиться электронная коллекция, и множество событий работы с данными коллекции. Для спецификации объектного представления ER-модели разработан метаязык для описания предметной области, а для описания жизненного цикла – язык шаблонов исходного кода.

В первой части процесса генерации строится внутреннее представление в виде набора объектов, каждому из которых сопоставляется множество параметров, используемых впоследствии при генерации исходного кода программ. Во второй части для каждого из событий жизненного цикла генерируются файлы исходного кода на целевом языке программирования. Каждый файл составлен из фрагментов, сформированных на основании соответствующего шаблона и объекта внутреннего представления. При этом внутреннее представление позволяет оценить, имеется ли возможность доступа ко всем данным системы и нет ли противоречий в представлении этих данных.

1. Андрианова А.А., Ицыксон В.М. Технология анализа исходного кода программного обеспечения и частичных спецификаций для автоматизированной генерации тестов. Системы и средства информации. 2014. Т. 24, Вып. 2. С. 99–113.
2. Балашова И.Ю., Волынская К.И., Макарычев П.П. Методы и средства генерации тестовых заданий из текстов на естественном языке // Модели, системы, сети в экономике, технике, природе и обществе. 2016. №1 (17).

3. Грибова В.В. Проблемно-независимый генератор текстов, управляемый онтологией // Проблемы управления. 2006. № 4. С. 36–42.
4. Камкин А.С. Генерация тестовых программ для микропроцессоров // Труды ИСП РАН. 2008. №2.
5. Леонтьев Д.В., Парахин Р.В., Тарасов Г.В., Харитонов Д.И. Моделирование предметной области для формирования электронных коллекций // Территория новых возможностей. Вестник Владивостокского государственного университета экономики и сервиса. 2018. Т. 10, № 2. С. 125–136.
6. Нархов К.Г. Генератор текста программ в исходном виде для систем реального времени // Программные продукты и системы. 2010. № 4.
7. Самохвалов Э.Н., Ревунков Г.И., Гапанюк Ю.Е. Генерация исходного кода программного обеспечения на основе многоуровнего набора правил // Вестник МГТУ им. Н.Э. Баумана. 2014. №5 (98).
8. Bennulf M., Svensson B., Danielsson F. Verification and deployment of automatically generated robot programs used in prefabrication of house walls. Procedia CIRP, 2018. № 72, P. 272–276.
9. Daniel D. McCracken and Edwin D. Reilly. Backus-Naur form (BNF). In Encyclopedia of Computer Science (4th ed.), Anthony Ralston, Edwin D. Reilly, and David Hemmendinger (Eds.). John Wiley and Sons Ltd., Chichester, UK. 2003. P. 129–131.
10. Diaz, Michel. Petri Nets: Fundamental Models, Verification and Applications. John Wiley and Sons. 2010.
11. Henrihs Gorskis, SQL Query Construction from Ontology Concept Descriptions, Information Technology & Management Science (RTU Publishing House). 2018. Vol. 12. P. 81–85.
12. Libin Hong, John H. Drake, John R. Woodward, Ender Ozcan, A Hyper-heuristic Approach to Automated Generation of Mutation Operators for Evolutionary Programming, 2017.
13. Leonard E.I., Heitmeyer C.L. Automatic Program Generation from Formal Specifications using APTS // Danvy O., Mairson H., Henglein F., Pettorossi A. (eds) Automatic Program Development. Springer, Dordrecht, 2008.
14. Paul Ammann and Jeff Offutt. Introduction to Software Testing (1 ed.). Cambridge University Press, New York, NY, USA, 2008.
15. Peter Pin-Shan Chen. 1976. The entity-relationship model-toward a unified view of data. ACM Trans. Database Syst. 1, 1. March, 1976. P. 9–36.
16. Smaragdakis Y. Program generators and the tools to make them. Lecture Notes in Computer Science. 2004. Vol. 3148. P. 19–20.

Транслитерация

1. Andrianova A.A., Icykson V.M. Tekhnologiya analiza iskhodnogo koda programmnogo obespecheniya i chastichnyh specifikacij dlya avtomatizirovannoj generacii testov. Sistemy i sredstva informacii. 2014. T. 24. Vyp. 2. P. 99–113.
2. Balashova I.Yu., Volynskaya K.I., Makarychev P.P. Metody i sredstva generacii testovyh zadanij iz tekstov na estestvennom yazyke // Modeli, sistemy, seti v ekonomike, tekhnike, prirode i obshchestve. 2016. №1 (17).
3. Gribova V.V. Problemno-nezavisimyj generator tekstov, upravlyaemyj ontologiej // Problemy upravleniya. 2006. № 4. P. 36–42.
4. Kamkin A.S. Generaciya testovyh programm dlya mikroprocessorov // Trudy ISP RAN. 2008. № 2.
5. Leont'ev D.V., Parahin R.V., Tarasov G.V., Haritonov D.I. Modelirovanie predmetnoj oblasti dlya formirovaniya elektronnyh kollekcij // Territoriya novyh vozmozhnostej. Vest-

nik Vladivostokskogo gosudarstvennogo universiteta ekonomiki i servisa. 2018. T. 10, № 2. P. 125–136.
6. Narhov K.G. Generator teksta programm v iskhodnom vide dlya sistem real'nogo vremeni. Programmnye produkty i sistemy. 2010. № 4.
7. Samohvalov E.N., Revunkov G.I., Gapanyuk YU.E. Generaciya iskhodnogo koda programmnogo obespecheniya na osnove mnogourovnego nabora pravi l// Vestnik MGTU im. N.E. Baumana. 2014. № 5 (98).

Для цитирования: Харитонов Д.И., Одякова Д.С., Леонтьев Д.В., Парахин Р.В. Генерация исходных кодов для тематических коллекций научных данных // Территория новых возможностей. Вестник Владивостокского государственного университета экономики и сервиса. 2019. Т. 11, № 4. С. 193–206.

For citation: Kharitonov D.I., Odyakova D.S., Leontiev D.V., Parakhin R.V. Source code generation for scientific data collections, *The Territory of New Opportunities. The Herald of Vladivostok State University of Economics and Service*, 2019, Vol. 11, № 4, pp. 193–206.

DOI dx.doi.org/10.24866/VVSU/2073-3984/2019-4/193-206

Дата поступления: 15.11.2019.