№ 1(8) 2014 ISSN 2309-8856

**Шахтинский институт (филиал)
федерального государственного бюджетного образовательного
учреждения высшего профессионального образования
"Южно-Российский государственный политехнический
университет (НПИ) имени М.И. Платова"**

**Донской государственный технический университет
Институт сферы обслуживания и предпринимательства
(филиал) ДГТУ в г.Шахты**

ОБЪЕКТНЫЕ СИСТЕМЫ – 2014

Материалы VIII Международной научно-практической конференции

**Россия, Ростов-на-Дону
10-12 мая 2014 г.**

№ 1(8) 2014 ISSN 2309-8856

**Шахтинский институт (филиал)
федерального государственного бюджетного образовательного
учреждения высшего профессионального образования
"Южно-Российский государственный политехнический
университет (НПИ) имени М.И. Платова"**

**Донской государственный технический университет
Институт сферы обслуживания и предпринимательства
(филиал) ДГТУ в г.Шахты**

ОБЪЕКТНЫЕ СИСТЕМЫ – 2014

Материалы VIII Международной научно-практической конференции

**Россия, Ростов-на-Дону
10-12 мая 2014 г.**

УДК 004 (05) **№ 1(8) 2014**
ББК 32.97-018 **ISSN 2309-8856**
О-29

Объектные системы – 2014: материалы VIII Международной научно-практической конференции (Ростов-на-Дону, 10-12 мая 2014 г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ (ф) ЮРГПУ (НПИ) им. М.И. Платова, 2014. – 108 с.

Материалы конференции посвящены принципам проектирования, реализации и сопровождения объектных систем и включают в себя освещение широкого круга проблем. Каждая статья, включённая в сборник, принята к печати на основании положительной рецензии членов организационного и программного комитетов.

Организаторы конференции

Шахтинский институт (филиал) федерального государственного бюджетного образовательного учреждения высшего профессионального образования "Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова"

Донской государственный технический университет Институт сферы обслуживания и предпринимательства (филиал) ДГТУ в г.Шахты

Информационные партнёры

Сообщество системных аналитиков

Теоретический и прикладной научно-технический журнал "**Информационные технологии**" с ежемесячным приложением

Ежемесячный научно-технический и производственный журнал "**Автоматизация в промышленности**"

Журнал "**Информационные технологии и вычислительные системы**"

Оргкомитет конференции

1. Прокопенко Николай Николаевич, д.т.н., проф., Заместитель директора по научно-исследовательской работе, Институт сферы обслуживания и предпринимательства (филиал) ДГТУ в г.Шахты, Россия, Шахты ***(председатель конференции)***
2. Олейник Павел Петрович, к.т.н., Системный архитектор программного обеспечения, Астон, Россия, Ростов-на-Дону ***(сопредседатель конференции)***
3. Божич Владимир Иванович, д.т.н., проф., Кафедра "Информационные системы и радиотехника", Институт сферы обслуживания и предпринимательства (филиал) ДГТУ в г.Шахты, Россия, Шахты
4. Сидельников Владимир Иванович, д.т.н., проф., Заведующий кафедрой "Экономика и прикладная математика", Педагогический Институт Южного Федерального университета, Россия, Ростов-на-Дону
5. Черкесова Эльвира Юрьевна, д.э.н., проф., Заведующая кафедрой "Управление персоналом, инновациями и качеством", Шахтинский институт (филиал), Южно-Российский государственный технический университет, Россия, Шахты
6. Михайлов Анатолий Александрович, д.т.н., проф., Кафедра "Автоматизированные системы управления", Южно-Российский государственный технический университет (Новочеркасский политехнический институт), Россия, Новочеркасск
7. Кравчик Владислав Георгиевич, к.т.н., доц., Кафедра "Энергетика и БЖД", Институт сферы обслуживания и предпринимательства (филиал) ДГТУ в г.Шахты, Россия, Шахты

Международный программный комитет конференции

1. Euclid Keramopoulos, Ph.D., Lecturer, Alexander Technological Educational Institute of Thessaloniki, Греция, Салоники
2. Piotr Habela, Ph.D., Assistant Professor, Polish-Japanese Institute of Information Technology, Польша, Варшава
3. Erki Eessaar, Ph.D., Associate Professor, Acting Head of the Chair, Faculty of Information Technologies: Department of Informatics, Tallinn University of Technology, Эстония, Таллин
4. German Viscuso, MSc in Computer Science, Marketing, Versant Corp., Испания, Мадрид
5. Кузнецов Сергей Дмитриевич, д.т.н., проф., Факультет вычислительной математики и кибернетики, МГУ имени М. В. Ломоносова, Главный научный сотрудник Института системного программирования РАН, член ACM, ACM SIGMOD и IEEE Computer Society, Россия, Москва
6. Шалыто Анатолий Абрамович, д.т.н., проф., лауреат премии Правительства РФ в области образования, Заведующий кафедрой "Технологии программирования", Санкт-Петербургский государственный университет информационных технологий механики и оптики, Россия, Санкт-Петербург
7. Кирютенко Александр Юрьевич, к.ф.-м.н., Директор по ИТ, Астон, Россия, Ростов-на-Дону
8. Галиаскаров Эдуард Геннадьевич, к.х.н, доц., Ивановский государственный химико-технологический университет, Россия, Иваново
9. Чекирис Александр Владимирович, Начальник отдела технического проектирования и НСИ, НИИЭВМсервис, Беларусь, Минск
10. Векленко Ирина Юрьевна, к.э.н., Системный аналитик, Россия, Черноголовка
11. Малышко Виктор Васильевич, к.ф.-м.н., доц., Факультет вычислительной математики и кибернетики, МГУ имени М. В. Ломоносова, Россия, Москва
12. Жилякова Людмила Юрьевна, к.ф.-м.н., доц., с.н.с., Институт проблем управления им. В.А. Трапезникова РАН, Россия, Москва
13. Шахгельдян Карина Иосифовна, д.т.н., Начальник информационно-технического обеспечения, Владивостокский государственный университет экономики и сервиса, Россия, Владивосток
14. Добряк Павел Вадимович, к.т.н., доц., Уральский государственный технический университет, Россия, Екатеринбург
15. Байкин Александр Сергеевич, Ведущий системный аналитик, Автомир, Россия, Москва
16. Аверин Алексей Иванович, Системный аналитик, Астон, Россия, Ростов-на-Дону
17. Лаптев Валерий Викторович, к.т.н., доц., Кафедра "Автоматизированные системы обработки информации и управления", Астраханский государственный технический университет, Россия, Астрахань
18. Ермаков Илья Евгеньевич, Генеральный директор, ООО "Метасистемы", Россия, Орёл
19. Иванов Денис Юрьевич, Консультант, Ай Ти Консалтинг, Россия, Санкт-Петербург

Рецензенты

Божич Владимир Иванович, Михайлов Анатолий Александрович, Шалыто Анатолий Абрамович, Малышко Виктор Васильевич, Добряк Павел Вадимович

Редакторы

Олейник Павел Петрович (***главный редактор***), Лаптев Валерий Викторович, Галиаскаров Эдуард Геннадьевич, Ермаков Илья Евгеньевич

ISBN 978-5-9903782-7-8
Статьи сборников конференции индексируются в РИНЦ
http://elibrary.ru/title_about.asp?id=48777

(с) **Объектные системы – 2014**, VIII Международная научно-практическая конференция, 2014
(с) **Коллектив авторов**, 2014

№ 1(8) 2014 ISSN 2309-8856

Shakhty Institute (branch) of
Platov South Russian State Polytechnic University (NPI)

Institute of Service and Business (branch)
Don State Technical University

Object Systems – 2014

Proceedings of the Eighth International Theoretical and Practical Conference

Edited by Pavel P. Oleynik

Rostov-on-Don, Russia
10-12 May 2014

№ 1(8) 2014 **ISSN 2309-8856**

Object Systems – 2014: Proceedings of the Eighth International Theoretical and Practical Conference (Rostov-on-Don, 10-12 May, 2014) / Edited by Pavel P. Oleynik. – Russia, Rostov-on-Don: SI (b) SRSPU (NPI), 2014. – 108 p.

The proceedings consist of papers which cover the topics of design work, implementation and maintenance of object systems, considering a broad range of problems. These papers were accepted to publish on the basis of the organization and program committee members reviews.

Conference Organizers

Shakhty Institute (branch) of Platov South Russian State Polytechnic University (NPI)

Institute of Service and Business (branch) Don State Technical University

Information Partners

Community of System Analysts

Journal "**Information Technologies**" with monthly supplement

Journal "**AUTOMATION IN INDUSTRY**"

Journal "**Information technologies and computer systems**"

Conference Committee

1. Nikolay N. Prokopenko, Doctor of Sciences, Professor, Deputy director for scientific research, Institute of Service and Business (branch) Don State Technical University, Russia, Shakhty ***(Chair of the conference)***
2. Pavel P. Oleynik, Ph.D., System Architect, Aston, Russia, Rostov-on-Don ***(Co-chair of the conference)***
3. Vladimir I. Bozhich, Doctor of Sciences, Professor, Department of Information Systems and Radio Engineering Institute of Service and Business (branch) Don State Technical University, Russia, Shakhty
4. Vladimir I. Sidelnikov, Doctor of Sciences, Professor, Head of the chair of Economics and Applied Mathematics, Southern Federal University, Pedagogical Institute, Russia, Rostov-on-Don
5. Elvira Yu. Cherkesova, Doctor of Sciences, Professor, Head of the chair of Human Resource Management, Innovation and Quality, Shakhty Institute (Branch) of South Russian State Technical University (Novocherkassk Polytechnic Institute), Russia, Shakhty
6. Anatoly A. Mikhailov, Doctor of Sciences, Professor, Department of Automated Control Systems, South Russian State Technical University (Novocherkassk Polytechnic Institute), Russia, Novocherkassk
7. Vyacheslav G. Krawczyk, Ph.D., Associate Professor of Energy and Life Safety, Institute of Service and Business (branch) Don State Technical University, Russia, Shakhty

International Program Committee

1. Euclid Keramopoulos, Ph.D., Lecturer, Alexander Technological Educational Institute of Thessaloniki, Greece, Thessaloniki
2. Piotr Habela, Ph.D., Assistant Professor, Polish-Japanese Institute of Information Technology, Poland, Warsaw
3. Erki Eessaar, Ph.D., Associate Professor, Acting Head of the Chair, Faculty of Information Technologies: Department of Informatics, Tallinn University of Technology, Estonia, Tallinn
4. German Viscuso, MSc in Computer Science, Marketing, Versant Corp., Spain, Madrid
5. Sergei D. Kuznetsov, Doctor of Sciences, Professor, Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University, Chief Scientist, Istitute for System Programming of Russian Academy of Science, ACM, ACM SIGMOD and IEEE Computer Society Member, Russia, Moscow
6. Anatoly A. Shalyto, Doctor of Sciences, Professor, Awarded by Russian State Government for achievements in education, Head of the chair of Programming Technologies, Saint-Petersburg State University of Information Technologies, Mechanics and Optics, Russia, Saint-Petersburg
7. Alexander Yu. Kiryutenko, Ph.D., CIO, Aston, Russia, Rostov-on-Don
8. Edward G. Galiaskarov, Ph.D., Associated Professor, Ivanovo State University of Chemistry and Technology, Russia, Ivanovo
9. Alexander V. Chekiris, Head of department of engineering design and normative-reference information, NIIEVMservice, Belarus, Minsk
10. Irina Yu. Veklenko, System Analyst, Russia, Chernogolovka
11. Victor V. Malyshko, Ph.D., Associated Professor of Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University, Russia, Moscow
12. Ludmila Yu. Zhilyakova, Ph.D., Senior Researcher, V.A. Trapeznikov Institute of Control Sciences, Russian Academy of Sciences, Russia, Moscow
13. Karina J. Shakhgeldyan, Doctor of Sciences, Head of IT-department, Vladivostok State University of Economics, Russia, Vladivostok
14. Pavel V. Dobryak, Ph.D., Associated Professor, Ural Federal University, Russia, Ekaterinburg
15. Alexander S. Baikin, Lead Systems Analyst, Avtomir, Russia, Moscow
16. Alexey I. Averin, Systems Analyst, Aston, Russia, Rostov-on-Don
17. Valery V. Laptev, Ph.D., Associated Professor of Automated Information Processing and Control System, Astrakhan State Technical University, Russia, Astrakhan
18. Ilya E. Ermakov, CEO, Metasystems Ltd., Russia, Orel
19. Denis Yu. Ivanov, Consultant, IT Consulting, Russia, Saint-Petersburg

Reviewers

Vladimir I. Bozhich, Anatoly A. Mikhailov, Irina Yu. Veklenko, Victor V. Malyshko, Pavel V. Dobryak

Editors

Pavel P. Oleynik ***(chief editor)***, Valery V. Laptev, Edward G. Galiaskarov, Ilya E. Ermakov

ISBN 978-5-9903782-8-5

(c) **Object Systems – 2014**, The Eighth International Theoretical and Practical Conference, 2014

(c) **Authors**, 2014

Содержание / Contents

Захарочкина А.П., Шведова И.П., Гринцова О.С. Организация удалённого рабочего места для выполнения научных проектов студентами в Шахтинском институте (филиале) ЮРГПУ (НПИ) им. М.И. Платова **9**

Шафоростова Е.Н., Ковтун Н.И., Михайлюк Е.А. Разработка CRM-системы для центра продаж и обслуживания корпоративных клиентов **11**

Дагаев Д.В. Не только потоки **15**

Хородов В.С. Проектирование многоагентной распределённой системы создания проектных решений с использованием структурно-функциональных лингвистических моделей **19**

Болдырев В.В., Горькавый М.А. Классификация управляющих воздействий в системах энергетического менеджмента организации, на основе объектно-ориентированного подхода **23**

Микляев И.А., Юрецкий А.П. Функциональная схема встроенного объектно-ориентированного интерактивного механизма конструирования страниц сайта на основе матричной универсальной объектно-реляционной базы данных **26**

Посконин А.В. Интеграция SQL-ориентированных СУБД и NoSQL-систем на уровне объектного отображения[1] **33**

Штанюк А.А. Объектно-ориентированные операционные среды **38**

Олейник П.П. Предметно-ориентированное проектирование структуры базы данных в понятиях метамодели объектной системы **41**

Мухутдинов Р.М., Шишкина В.В., Грегер С.Э. Визуализация онтологических моделей и статических данных **46**

Сергеев Е.В., Горькавый М.А. Разработка модуля управления уровнем освещённости локальных зон многофункционального помещения на основе объектно-ориентированного подхода **49**

Мясникова Н.А., Шепилов В.А. Сериализация и десереализация объектов Python в YAML-формат **52**

Хоченков А.Е., Галиаскаров Э.Г. Автоматизированная система сбора данных физического эксперимента **57**

Прилепко М.А. Технология разработки системы автоматизации проектирования “Network Lab” **63**

Кураков Ю.И., Олейник П.П. Проектирование ядра информационной системы экономического производственно-энергетического кластера угольной промышленности **67**

Никулин В.В., Пахмутов А.В., Грегер С.Э. Моделирование учебного процесса с использованием онтологии жизненного цикла **73**

[1] Статья рекомендована к опубликованию в журнале "**Информационные технологии**"

Содержание / Contents

Шадрина Е.В., Назаров М.А., Грегер С.Э. Управление структурой и визуальным отображением системы управления учебным процессом на основе семантической сети **76**

Салибекян С.М., Халькина С.Б., Тиновицкий К.Д. Объектно-атрибутный подход для семантического анализа естественного языка[1,2] **80**

Олейник П.П. Тестовая модель для обучения проектированию объектно-ориентированных баз данных **86**

Маликова В.И., Олейник П.П. Организация портала научного сотрудничества **89**

Лаптев В.В., Грачев Д.А. Семантическая модель программы и ее реализация[3] **92**

IX Международная научно-практическая конференция **"Объектные Системы – 2015"** **99**

Для заметок / Notes **102**

[1] Статья рекомендована к опубликованию в журнале "**Информационные технологии и вычислительные системы**"

[2] Исследование осуществлено в рамках Программы фундаментальных исследований НИУ ВШЭ в 2014 году

[3] Статья рекомендована к опубликованию в журнале "**Информационные технологии**"

УДК 004.04

ОРГАНИЗАЦИЯ УДАЛЁННОГО РАБОЧЕГО МЕСТА ДЛЯ ВЫПОЛНЕНИЯ НАУЧНЫХ ПРОЕКТОВ СТУДЕНТАМИ В ШАХТИНСКОМ ИНСТИТУТЕ (ФИЛИАЛЕ) ЮРГПУ (НПИ) ИМ. М.И. ПЛАТОВА

Захарочкина Анна Петровна, студентка, кафедра "Естественнонаучные дисциплины", ШИ (ф) ЮРГПУ (НПИ) им. М.И. Платова, Россия, Шахты
Шведова Ирина Петровна, студентка, кафедра "Естественнонаучные дисциплины", ШИ (ф) ЮРГПУ (НПИ) им. М.И. Платова, Россия, Шахты
Гринцова Ольга Сергеевна, студентка, кафедра "Естественнонаучные дисциплины", ШИ (ф) ЮРГПУ (НПИ) им. М.И. Платова, Россия, Шахты

Привлечение студентов к научной деятельности является неотъемлемой обязанностью заведующего любой кафедрой. В настоящей статье представлена методика организации научной деятельности студентов в Шахтинском институте (филиале) ЮРГПУ (НПИ) им. М.И. Платова, разработанная авторами на кафедре "Естественнонаучные дисциплины" и используемая при реализации различных ИТ-проектов.

При организации научной деятельности студентов формулировались следующие критерии оптимальности (КО):

1. Привлекать студентов с первого курса обучения. Т.е. предложенная методика не должна иметь высокий уровень вхождения в задачу.
2. Предоставить унифицированный доступ для всех научных проектов. Такой подход позволит увеличить возможности обмена информацией между студентами.
3. Сократить до минимума программное обеспечение, установленное на компьютерах студентов и требуемое для подключения к проектам.
4. Предоставить возможность использования различных гаджетов и устройств: от персональных компьютеров до планшетов и мобильных телефонов.

Для графического отображения принципов организации научной деятельности используется диаграмма развертывания унифицированного языка моделирования UML [1], представленная на рисунке 1.

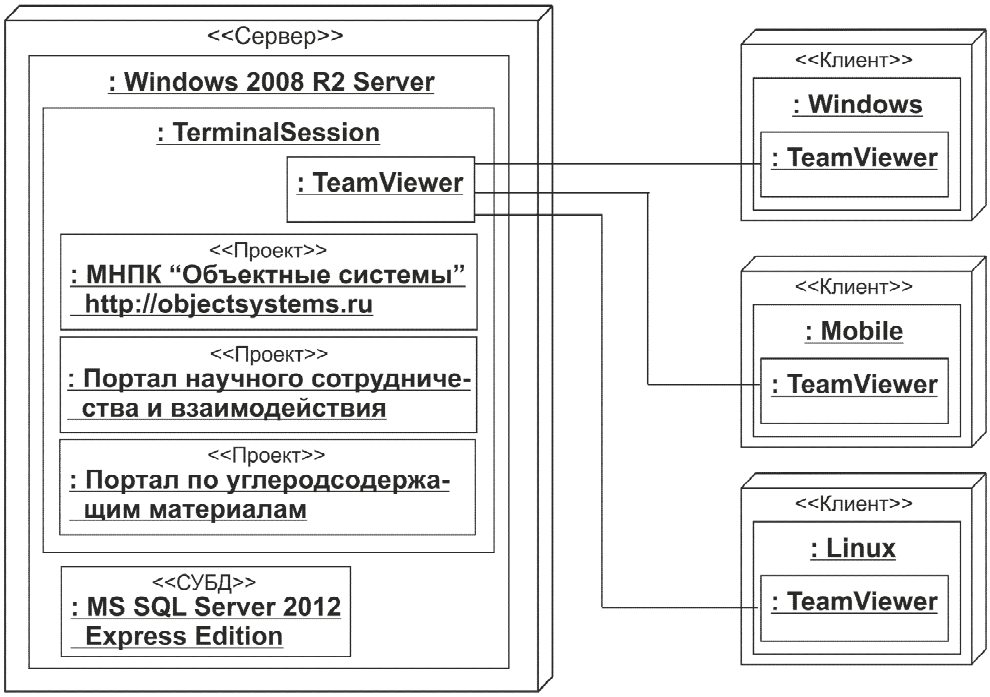

Рис. 1 – Диаграмма развёртывания языка UML

Из рисунка видно, что реализована классическая двухзвенная архитектура, содержащая два узла развертывания: клиентский узел и серверный.

Основным является серверный узел, который представляет собой аппаратно-программную платформу, работающую в режиме 24X7 в течение всего времени. Первая проблема, которая возникла при организации сервера, – это проблема подключения к нему через глобальную сеть Интернет. Для этого необходимо наличие уникального адреса, по которому он был бы доступен клиентам. Общепринятой практикой в таком случае является приобретение выделенного статического IP-адреса для сервера.

В нашем случае такой подход был исключен по следующим причинам:

1. Приобретение выделенного статического IP-адреса требует ежемесячных постоянных затрат.
2. На сервере установлен Web-сервер, для которого уже используется статический IP-адрес, а управление несколькими адресами является не рациональным.

В связи с описанными проблемами было принято решение об использовании промежуточного программного обеспечения для коммуникации клиента и сервера. Выбор пал на продукт TeamViewer – решение "всё в одном" для удалённого доступа и поддержки через Интернет [2].

Основная идея использования TeamViewer 9 заключается в установке продукта на серверную операционную систему и добавлении роли "Терминальный сервер". Терминальный сервер был организован на базе операционной системе Microsoft Windows 2008 R2 Server, лицензия на которую имеется в организации.

Для каждого пользователя заводится отдельная учетная запись и предоставляется возможность открытия терминальной сессии. В результате студент подключается к собственной сессии и может работать, не пересекаясь с остальными участниками проекта.

Для подключения к терминальному серверу используется некоммерческая версия программы TeamViewer 9, которая доступна для различных аппаратно-программных платформ. В результате студенту достаточно установить программу TeamViewer на своё устройство. При установке на терминальный сервер каждому пользователю генерируется собсвенный идентификатор TeamViewer.

Если рассматривать аппаратную составляющую сервера, то она относительно проста. В сервер установлен винчестер Intel 520 SSD объёмом 120 Gb с 8 Gb оперативной памятью стандарта DIMM DDR3 PC12800. Описанная конфигурация успешно протестирована на одновременной работе 50 пользователей.

Рассмотрим соответствие предложенного решения задачи выделенным в начале статьи критериям оптимальности. Имеется полное соответствие требованиям КО1, т.к. от студента требуется установить лишь одну программу TeamViewer. С этой задачей успешно справляется каждый первокурсник. Следствием предложенного подхода является автоматическое выполнение требований КО3. Отметим, что TeamViewer предоставляет некоммерческую версию продукта для различных платформ (см. рис. 1), поэтому отсутствуют требования к клиентским устройствам и выполнены условия КО4. Как отмечалось ранее, каждому пользователю отводится отдельная терминальная сессия, т.е. предоставляется унифицированный доступ для всех научных проектов, что свидетельствует о выполнении требованиям КО2.

Перейдем к рассмотрению научных проектов, в которых участвуют студенты кафедры "Естественнонаучные дисциплины". В настоящий момент их 3:

1. Международная научно-практическая конференция "Объектные системы" (objectsystems.ru);
2. Портал научного сотрудничества и взаимодействия;
3. Портал по углеродсодержащим материалам.

С 2010 года в нашем ВУЗе регулярно проводится международная научно-практическая конференция "Объектные системы", которая посвящена принципам проектирования, реализации и сопровождения объектных систем и включает в себя обсуждение широкого круга проблем. По окончании работы конференции публикуется сборник научных трудов авторов с присвоением ему ISSN, ISBN, УДК и ББК кодов. Статьи сборников индексируются

в РИНЦ. Лучшие, по мнению рецензентов, доклады рекомендованы к бесплатной публикации в журналах, входящих в перечень ВАК. Организация и проведения конференции – объемная задача, к выполнению которой привлекаются наши студенты. Для упрощения процесса проведения МНПК "Объектные системы" разработан программный комплекс, в котором каталогизируется информация обо всех авторах, написанных статьях, рецензиях и т.п. [4-5].

За долгие годы существования нашей кафедры у сотрудников и студентов возникли обширные научные связи, которые основаны на принципах взаимодействия и взаимопомощи. В настоящее время ведутся разработки нового единого информационного ресурса, реализованного в виде портала научного сотрудничества и взаимодействия.

Самым последним научным проектом, который выполняется студентами, является проект по созданию портала, содержащего информацию и технические характеристики углеродсодержащих материалов. Ростовская область – угледобывающая часть России. Поэтому создание подобного портала является своевременным и необходимым решением насущных задач угледобывающей отрасли нашей страны и ближнего зарубежья. На портале будет представлена информация обо всем спектре углеродсодержащих материалов от классификаций каменного угля и до материалов, получаемых из полезных ископаемых. Предстоит выполнить огромную работу по разработке структуры базы данных и заполнения её уникальной информацией, имеющейся сейчас только на бумажных носителях.

Для сокращения времени на выполнение задач и увеличения производительности труда на сервере установлено собственное разработанное программное обеспечение, использующее в качестве СУБД Microsoft SQL Server 2012 Express Edition. Бесплатной версии вполне достаточно для реализации научных проектов нашего ВУЗа. В последнее время всё большую популярность получают облачные вычисления. В связи с этим, в будущем планируется перенести все проекты в отдельное облако.

Литература

1. Новиков Ф.А., Иванов Д.Ю. Моделирование на UML. Теория, практика, видеокурс. – СПб.: Профессиональная литература, Наука и Техника, 2010. – 640 с.
2. TeamViewer – бесплатный удалённый доступ и общий доступ к рабочему столу через Интернет, www.teamviewer.com
3. Международная научно-практическая конференция "Объектные системы", objectsystems.ru
4. Олейник П.П., Игумнов Е.А., Свечкарёв Е.А. Опыт проектирования информационной системы для каталогизирования научных работ при проведении международных конференций // Объектные системы – 2010: материалы II Международной научно-практической конференции. Россия, Ростов-на-Дону, 10-12 ноября 2010 г., Ростов-на-Дону, 2010. – С. 48-51.
5. Олейник П.П., Игумнов Е.А., Свечкарёв Е.А. Реализация модуля рецензирования в информационной системе проведения научных конференций // Объектные системы – 2011: материалы III Международной научно-практической конференции (Ростов-на-Дону, 10-12 мая 2011 г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону, 2011. – С. 26-29.

УДК 004.02

РАЗРАБОТКА CRM-СИСТЕМЫ ДЛЯ ЦЕНТРА ПРОДАЖ И ОБСЛУЖИВАНИЯ КОРПОРАТИВНЫХ КЛИЕНТОВ

Шафоростова Елена Николаевна, к.п.н., доцент, Старооскольский технологический институт НИТУ «МИСиС», Россия, Старый Оскол, shaf-elena@yandex.ru

Ковтун Нелли Игоревна, старший преподаватель, Старооскольский технологический институт НИТУ «МИСиС», Россия, Старый Оскол

Михайлюк Екатерина Андреевна, старший преподаватель, Старооскольский технологический институт НИТУ «МИСиС», Россия, Старый Оскол

В настоящее время невозможно представить процесс управления производством, выполняемый без участия информационных систем, которые облегчают и улучшают технологические процессы и качество обработки информации.

Грамотно обработанная и систематизированная информация является в определенной степени гарантией эффективного управления производством. Напротив, отсутствие оперативных и достоверных данных может привести к неверному управленческому решению и, как следствие, к серьезным убыткам. Объединенная компания ОАО «Ростелеком» обладает полным комплексом государственных лицензий, позволяющих оказывать широкий спектр телекоммуникационных услуг во всех регионах России.

Разработка информационной системы учета клиентов (CRM – система) с целью повышения производительности труда активных продавцов, которые занимаются поиском и привлечением клиентов, является актуальной задачей. Для решения поставленной задачи необходимо разработать CRM – систему, целью которой является автоматизация рутинных операций деятельности активных продавцов, что позволит им сконцентрироваться на творческом процессе человеческого общения и более эффективно заниматься поиском, привлечением и удержанием прибыльных клиентов [1].

Проведем анализ необходимых информационных потоков, для решения поставленной задачи. В качестве входной информации будет использовать следующие данные:

- информация о клиенте (Наименование организации; Город; Улица; Дом; Квартира/офис; Контактное лицо; Телефон; Положение на карте);
- служебная информация (№ договора; Лицевой счет; Дата; Услуга; Техническая возможность; Заметки).

Реализация этой подзадачи позволит:

- обеспечить удобный и эффективный ввод информации даже неквалифицированным пользователем;
- создать максимально комфортные условия работы;
- повысить достоверность и полноту информации.

При этом выполняются следующие функции:

- вывод на экран форм для ввода данных;
- считывание данных, вводимых с клавиатуры;
- контроль вводимой информации;
- занесение считанных данных в таблицы БД.

Следующей подзадачей является процесс обработки полученной информации, с целью преобразования информации в БД для ее дальнейшего хранения и использования.

Реализация этой подзадачи позволит:

- обеспечить удобный для дальнейшего использования вид хранимой информации;
- снизить трудоемкость при поиске в БД конкретной информации, необходимой для использования;
- обеспечить возможность редактирования и удаления ранее введенной информации.

В процессе реализации данной подзадачи выполняются следующие функции:

- вывод на экран форм;
- считывание ключевых данных;
- контроль вводимой информации;
- поиск в базе данных записи с заданными ключевыми данными;
- редактирование найденной записи и при необходимости ее удаление;
- обновление данных в таблицах БД.
- подготовка, формирование и выдача информации.

Следующей подзадачей является вывод информации, с целью формирования по требованию пользователя и выдачи на экран или печатающее устройство документов и

отчетов, необходимых в существующей системе документооборота, с необходимой периодичностью.

Реализация данной подзадачи позволит:

- оперативно формировать и выдавать на экран терминала или печатающее устройство необходимые пользователю документы или отчеты, существующие в системе документооборота, в любой момент времени;
- повысить эффективность анализа оперативной ситуации, сложившейся в отделе к данному моменту;
- автоматизировать формирование документов и справок.

Ведение нормативно-справочной информации является очень важным моментом для любого предприятия. Назначение справочников заключается в систематизации хранящейся информации, увеличении скорости ввода информации и уменьшении количества возможных ошибок. Это достигается тем, что все новые данные определенного типа вносятся в справочник. Повторное использование одного и того же названия приводит фактически к многократному использованию одной и той же записи из таблицы справочника [2].

Таким образом, для эффективной работы необходимо создать следующие справочники:

1. Справочник «Услуги», который содержит список оказываемых услуг:
   - id услуги;
   - название услуги.
2. Справочник «Города»:
   - id города;
   - название города.
3. Справочник «Улицы»:
   - id улицы;
   - название улицы.

При построении базы данных необходимо выделить следующие сущности (рис. 1):

- «Клиент»;
- «Услуга»;
- «Город»;
- «Улица»;
- «АТС»;
- «Координаты».

Центральной информационной сущностью БД является сущность «Клиент», так как она формируется на основе всей остальной информации.

Несколько клиентов используют одну услугу, поэтому сущность «Клиент» связана отношением «Использует» с сущностью «Услуга» и имеет связь «многие к одному».

Несколько клиентов находятся в одном городе, поэтому сущность «Клиент» связана отношением «Находится» с сущностью «Город» и имеет связь «многие к одному».

В одном городе содержится несколько улиц, поэтому сущность «Город» связана отношением «Содержит» с сущностью «Улица» и имеет связь «один ко многим».

Несколько клиентов имеют одни координаты, поэтому сущность «Клиент» связана отношением «Имеет» с сущностью «Координаты» и имеет связь «многие к одному».

Нескольким клиентам принадлежит одна АТС, поэтому сущность «Клиент» связана отношением «принадлежит» с сущностью «АТС» и имеет связь «многие к одному».

Даталогическая модель данных (рис. 2) для разрабатываемой системы построена с использованием MySQL Workbench 5.2 CE, поскольку она является бесплатной программой для проектирования, моделирования, создания баз данных и настройки MySQL-сервера и позволяет визуально отображать структуру, состояние и модель сервера баз данных на основе простого подключения к нему.

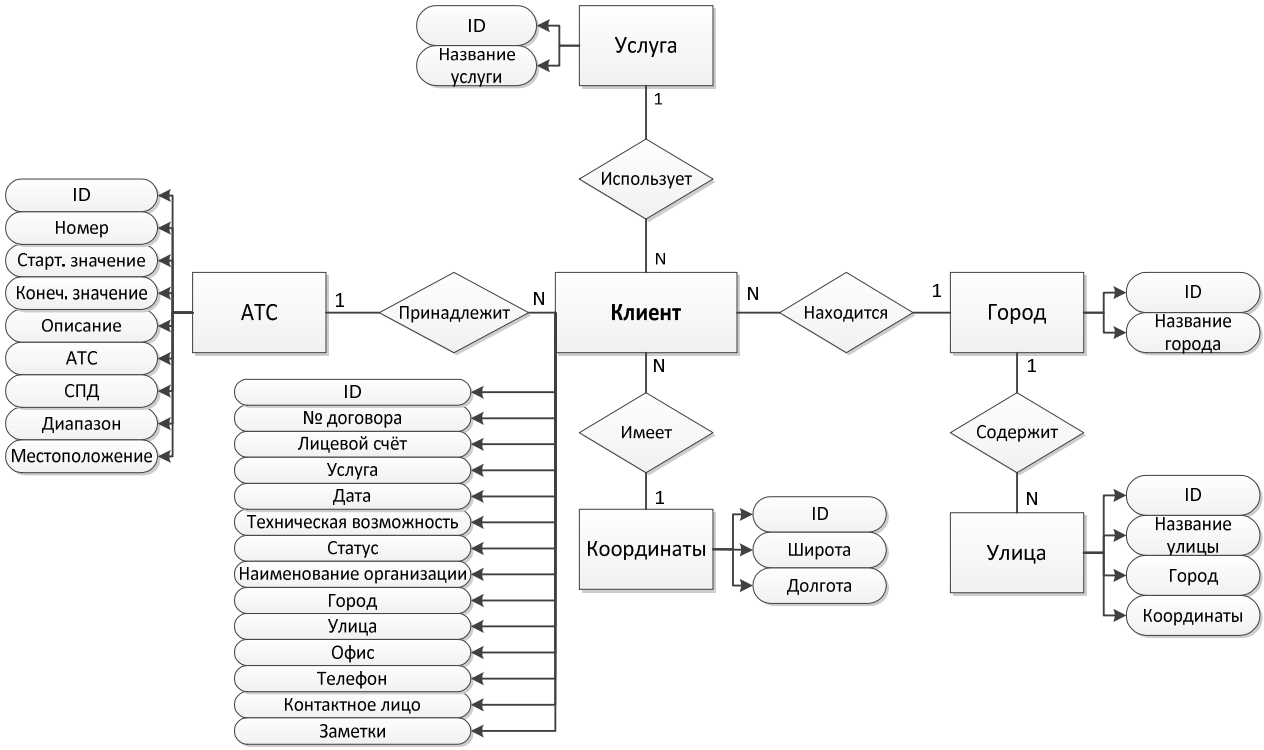

Рис. 1 – Инфологическая модель данных

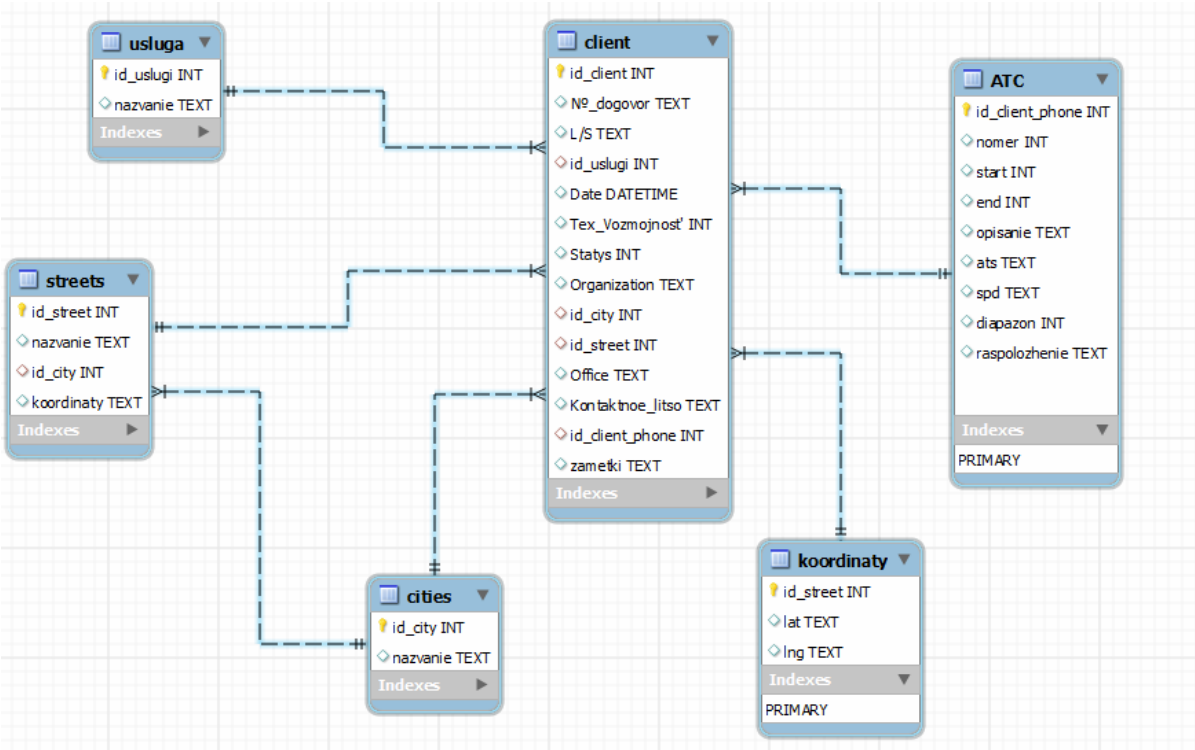

Рис. 2 – Даталогическая модель

Все таблицы необходимо нормализовать во избежание избыточности данных, аномалий обновления, удаления и ввода; для поддержания целостности данных [3].

Для практической реализации поставленной задачи создания CRM-системы необходимо разработать соответствующее информационное обеспечение. При анализе веб-проектов выявлено, что многие из них разработаны на PHP, с использованием СУБД MySQL и функционируют на веб-сервере Apache2,2 и ОС Linux. На ОАО «Ростелеком» уже установлен данный набор. Для администрирования MySQL на сервер необходимо установить phpmyadmin: sudo apt-get install phpmyadmin.

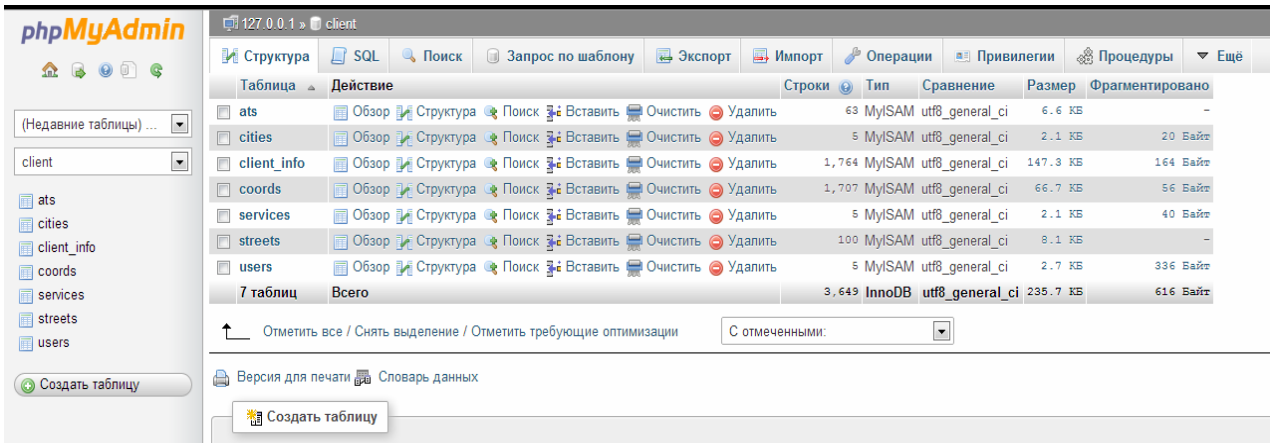

Рис.3 – phpMyAdmin

Разработку программного кода и визуальное проектирование можно выполнить с использованием Nootepad++ и ознакомительной версии Adobe Dreamweaver CS6 (HTML-редактор). При редактировании Adobe Dreamweaver показывает готовую страницу в режиме WYSIWYG (что видишь, то и получишь), что является достаточно удобным и быстрым для создания web-интерфейса. Данные программные продукты полностью совместимы между собой и с установленной операционной системой [4].

Разрабатываемая CRM-системы позволит: повысить производительность труда активных продавцов, создать базу данных с разграничением прав доступа. Внедрение информационной системы значительно облегчает данную работу, поскольку создается база данных, которая позволит оперативно отражать информацию. Также значительно облегчается сам процесс изменения и ведения бумажной документации, поскольку основные элементы операций автоматизированы и выполняются по достаточно несложному алгоритму, в результате чего экономится время, человеческие ресурсы, оперативно поступает информация и повышается качество работы в целом.

Литература

1. Keith Brandt. When Bad Harmonics Happen to Good People. – OSP. November 2007.P.- 16.
2. Горев А. Эффективная работа с СУБД – М.: Инфра, 2000 г., 504 с.
3. Кузнецов С.Д. СУБД (системы управления базами данных) и файловые системы.- М: Майор, 2001 г
4. Никифоров С.В. Введение в сетевые технологии: Элементы применения и администрирования сетей: Учеб. пособие. – М.:Финансы и статистика, 2010. – 224 с

УДК 004.04

НЕ ТОЛЬКО ПОТОКИ

Дагаев Дмитрий Викторович, Директор по АСУТП, ОАО «ДЖЭТ», Россия, Москва,
dvdagaev@mail.ru

Задачи и сопрограммы

При разработке серверных приложений часто встает вопрос необходимости одновременной работы и взаимодействия нескольких подзадач. Это сродни многозадачности, в [1] говорится о "разновидности процесса внутри самого процесса". Как решения для серверной задачи предлагаются 3 варианта:

- многопоточный процесс;
- однопоточный процесс;
- параллельная работа с неблокирующими системными вызовами.

Там, где однопоточный процесс с блокирующими системными вызовами невозможен, обычно используется многопоточное программирование. Мы же будем рассматривать последний случай, который предполагает несколько нитей вычислений с сохраняемым состоянием у каждого, и события, по которым осуществляются переходы состояний. В качестве реализации предлагаются:

- задачи (Tasks), в которых нужно делать функцию перехода состояний в стиле конечного автомата, и возвращающие управление;
- сопрограммы (Coroutines), позволяющие выходить из произвольной точки кода процедуры и возвращаться на следующий за ней оператор.

Для каждого объекта-задачи вводится состояние и функция перехода, которая для простого примера печати натуральных чисел выглядит так:

```
PROCEDURE Do (t: Task);
BEGIN IF t.count < t.max THEN INC(t.count); Log.Int(t.count); Log.Ln; END
END Do;
```

Функция должна быть написана таким образом, чтобы при периодическом вызове изменять состояние `t.count` и возвращать управление.

Вариант с объектом-сопрограммой более универсален:

```
PROCEDURE Do (t: Cotoutine); VAR j: INTEGER;
BEGIN j := 0; WHILE TRUE DO INC(j); Log.Int(j); Log.Ln; Co.Yield END
END Do;
```

Функция содержит только точку выхода `Co.Yield` и возвращается при следующем вызове на следующий за ней оператор.

Использование сопрограмм имеет долгую историю. По утверждению Кнута [2], "подпрограмма является частным случаем сопрограммы". Сопрограммы широко использовались в ассемблерных проектах, Вирт ввел сопрограммы в язык Модула-2. Работа подпрограммы определяется более строгой дисциплиной LIFO, при которой выделяется стековая область для локальных переменных каждого экземпляра подпрограммы (детально: записывает в стек адрес следующей команды, переходит по адресу процедуры, выделяет в стеке место для локальных переменных). При выходе из подпрограммы эта область освобождается (помещая в стек возвращаемое значение и переходя к следующей команде).

В сопрограммах такой дисциплины нет, для их использования нужны 3 свойства [3]:

- локальные данные сохраняются между успешными вызовами;
- выполнение приостанавливается в определенных точках разрыва, при дальнейшей передаче управление переходит на следующий оператор;
- управление может передаваться от одной сопрограммы к другой, при этом вызывающая сопрограмма приостанавливается. Последнее свойство запрещено использовать для решения задач разделения времени, чтобы обеспечить герметичность взаимодействия.

Оператор `Yield` осуществляет как раз этот выход из процедуры с последующим возвратом на следующий за ним оператор. Это достигается путем отдельного сохранения всего стека выполнения каждой сопрограммы в предварительно отведенной памяти, а также состояния регистров (указателя стека, счетчика команд, и др.). При вызове `Yield` происходит такой переход с переключением контекстов, но сопрограммы считаются легковеснее потоков, т.к. последние, как минимум, имеют дополнительные издержки на штатный планировщик.

Для управления задачами и сопрограммами потребовалось реализовать планировщик на уровне пользовательского процесса, который и осуществляет механизмы переключения.

Объектная модель планировщика

Разработанная модель берет свое начало в [4], при этом пользовательская задача наследуется от Task и реализует состояния и функцию перехода `Do`, либо пользовательская

сопрограмма наследуется от Coroutine и реализует вызываемую планировщиком функцию `Do` с точками разрыва. В соответствие с заложенными принципами, пользовательский тип наследуется от типа, определенного в Tasks, и реализуется виртуальная функция `Do`. Т.е. есть наследование типов и нет наследования реализации.

Построение основано на тройке классов Scheduler/Task/Dispatcher, предложенной в [4] как вариация обобщенной тройки Carrier/Rider/Link (другой вариацией является известная Модель/Вид/Контроллер). При этом иерархии классов Task-Coroutine-UserProgram (возможно, и Task-UserProgram) и Dispatcher-S_Dispatcher+Scheduler-S_Scheduler развиваются независимо.

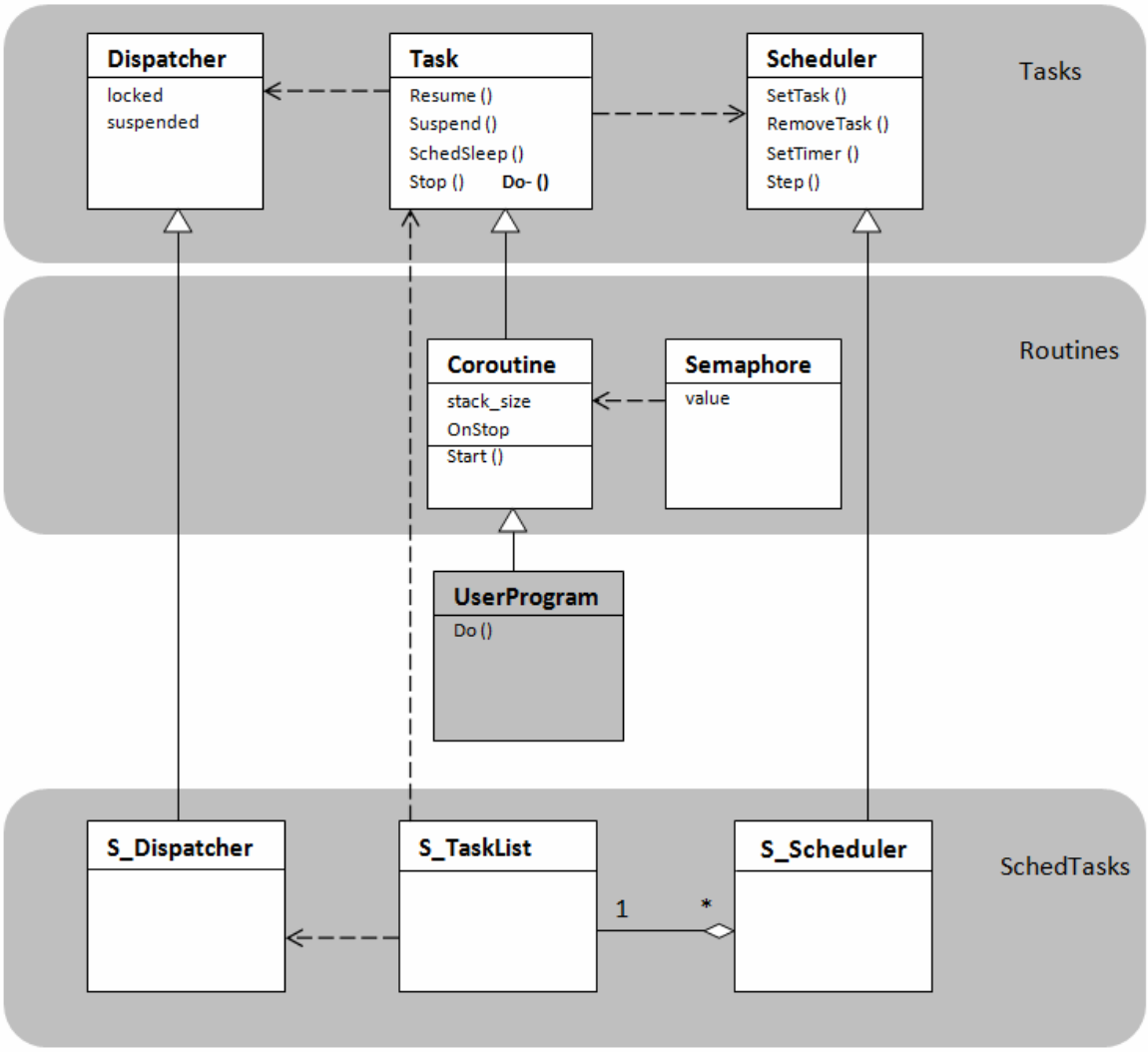

Рис. 1 – Объектная модель планировщика

Первая группа отвечает за пользовательскую реализацию. В соответствии с принципом отделения абстракции от реализации пользовательская программа наследуется от типа Coroutine и остается одинаковой, несмотря на разные реализации сопрограммы на 2 компиляторах для 2 ОС каждый. Помимо реализации метода `Do` нужно также иметь в виду, что переходы на другие сопрограммы в режиме планировщика запрещены на уровне рантайма, а управление поведением сопрограмм (скажем, для защиты общих ресурсов) осуществляется с помощью семафоров и методов Resume, Suspend, SchedSleep.

Вторая группа S_Scheduler+S_Dispatcher представляет собой реализацию абстрактных типов Scheduler и Dispatcher, которая работает со списком задач S_TaskList, выбирая адрес, куда передается управление. Реализуется алгоритм циклического планирования с двумя уровнями приоритета. В зависимости от параметров планировщика, определяющих квоту времени, переход в точках разрыва может не осуществляться, если квота не выбрана.

Следует сравнить механизмы семафоров сопрограмм, с одной стороны, и считающиеся более структурированными мониторы Хоара-Хансена [1] и активные объекты, с другой [5]. Отсутствие вытесняющей многозадачности меняет поведение семафоров, т.к. вся сопрограмма превращается в «критическую область» в смысле Хансена или в эксклюзивную секцию активного объекта. Семафор определяет точку разрыва защищенной области, в которой сопрограмма будет находиться в состоянии активного ожидания:

```
BEGIN Do1; Co.SemDown(sem); Do2 END
```

Аналогичная секция активного объекта на языке Активный Оберон [5] будет выглядеть:

```
BEGIN {EXCLUSIVE} Do1; AWAIT(sem>0); DEC(sem); Do2 END
```

Следует заметить, что активные объекты и мониторы разрабатывались на замену менее структурированным средствам синхронизации, в т.ч. семафорам. Но для сопрограмм эта неструктурированность уходит, причем это решается без дополнительных языковых средств.

Сравнение реализаций

Для языков Оберон-семейства BlackBox и XDS и операционных систем Linux и Windows сравнивались варианты реализации.

XDS подключает одинаковый для Linux и Windows библиотечный модуль COROUTINES, основные функции которого – старт и переключение:

```
COROUTINES.NEWCOROUTINE(RunProc, stack, ssize, coroutine); (* старт *)
COROUTINES.TRANSFER(from, coroutine); (* переключение *)
```

BlackBox использует функции работы с Fibers библиотеки win32:

```
primary := WinApi.ConvertThreadToFiber(0);
fiber := WinApi.CreateFiber(ssize, RunProc, coroutine); (* старт *)
WinApi.SwitchToFiber(fiber); (* переключение *)
```

BlackBox для Linux использует для аналогичных целей контексты, подключая модуль CA как библиотеку libc CA[“libc.so.6”]:

```
context.uc_stack.ss_sp := stack;
context.uc_stack.ss_size := ssize;
rc := CA.makecontext(context, RunProc, 3, context
      , tmp_context, coroutine); (* старт *)
rc := CA.swapcontext(tmp_context, context) (* переключение *)
```

Были оценены времена переключений между сопрограммами по сравнению с временами переключения между потоками в реализации на C в тестовом примере с числом задач = 100.

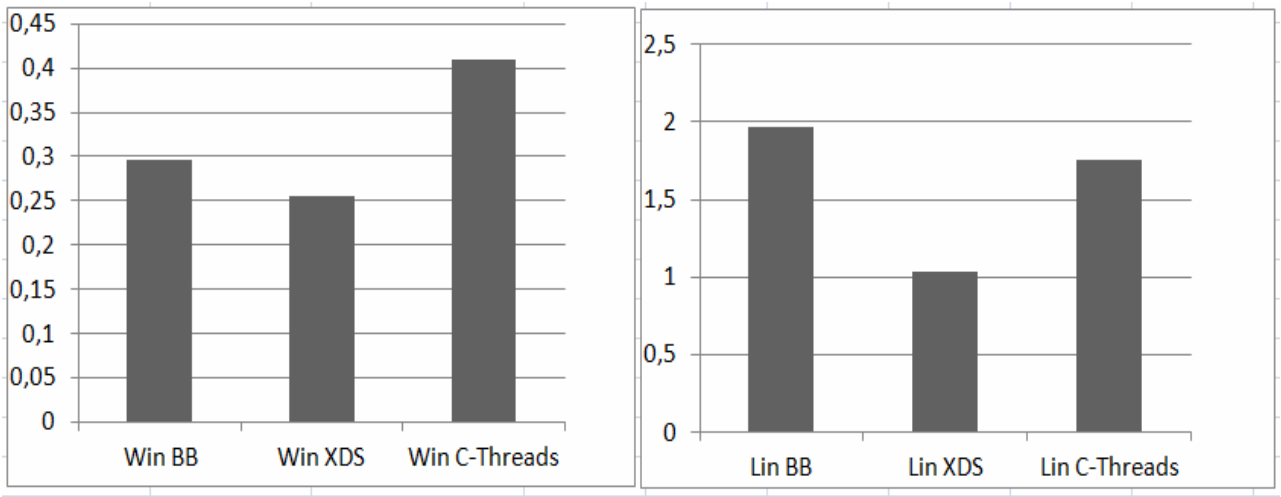

Рис. 2 – Среднее время переключения между задачами, мкс

Приведенные диаграммы носят в большей степени качественный характер, т.к. реальные характеристики сильно зависят от оборудования, на котором производится тестирование. В нормальном случае переключение между заведомо более «тяжеловесными» потоками должно занимать большее время, поэтому дополнительные 40-70% времени для потоков представляются разумной цифрой. Только времена по переключению контекстов BlackBox для Linux отличаются в худшую сторону, в связи с чем можно посетовать на реализацию библиотеки контекстов в Linux или перейти на более низкий, ассемблерный уровень.

В качестве рекомендаций по применению следует иметь в виду несколько моментов. Эти данные соответствуют однопроцессорному аппаратному обеспечению, многопоточные приложения, в отличие от сопрограмм, распределяются по аппаратуре. Также, несмотря на полную защищенность Оберонов, их сборщики мусора не отслеживают локальные стеки сопрограмм, поэтому указатели как переменные в локальных стеках не должны оставаться последними адресатами памяти, что-то должно быть в «куче» или основной программе. Зато такой подход дает более гибкие способы управления планированием.

Литература

1. Э.Таненбаум. Современные Операционные Системы. 3-е изд. 2012.
2. Искусство программирования, том 1. Основные алгоритмы = The Art of Computer Programming, vol.1. Fundamental Algorithms. — 3-е изд. — М., 2006
3. George Frasier “Cross-Platform Coroutines in C++”, Dr.Jobbs, March 01, 2001
4. C.Szyperski "Insight-EthOS: An Object-Orientation in Operating Systems", 1992.
5. P.J. Muller. The Active Object System – Design and Multiprocessor Implementation. PhD thesis, ETH Zurich, 200

УДК 004.896

ПРОЕКТИРОВАНИЕ МНОГОАГЕНТНОЙ РАСПРЕДЕЛЁННОЙ СИСТЕМЫ СОЗДАНИЯ ПРОЕКТНЫХ РЕШЕНИЙ С ИСПОЛЬЗОВАНИЕМ СТРУКТУРНО-ФУНКЦИОНАЛЬНЫХ ЛИНГВИСТИЧЕСКИХ МОДЕЛЕЙ

Хородов Виталий Сергеевич, аспирант кафедры вычислительная техника, Ульяновский государственный технический университет, Россия, г. Ульяновск, v.khorodov73@gmail.com

Введение

Современной парадигмой, нашедшей широкое применение в практике проектирования сложных технических объектов, является smart environment («умная» среда). При этом большое внимание уделяется построению среды взаимодействия, которая способствует совместной работе проектировщиков [1]. Одной из основных тенденций в развитии современных САПР является распределённая реализация проектных процедур проектировщиками, которые находятся на удалённом расстоянии друг от друга.

В настоящее время одним из факторов, характеризующих возрастание сложности процесса проектирования, является объем создаваемого кода на Hardware Description Language (HDL) языке и количество логических вентилей в проектируемых схемах.

Реализация связующего программного обеспечения (middleware level) – необходимый этап для организации взаимодействия между проектировщиками. Для размещений связующего ПО можно использовать облачные технологии, которые должны способствовать реализации системы распределённого проектирования.

В статье предлагается подход к построению системы распределённого проектирования, которая позволит создавать проектные решения с использованием структурно-функциональных лингвистических моделей (СФЛМ), в которой используется агентно-ориентированная технология. Обосновывается выбор агентно-ориентированной

технологии создания архитектуры системы и описывается концепция разрабатываемого метода автоматизации проектирования.

Система работает со структурно-функциональными лингвистическими моделями, которые хранятся в базе знаний. Под СФЛМ понимаются объекты, представленные на языке типа VHDL, состоящие из структурной и функциональной части.

Структурное описание представляет собой иерархическую архитектуру связанных компонентов. Функциональное (поведенческое) описание представляет собой архитектуру, состоящую из множества параллельных регистровых операций, каждая из которых управляется вентильными сигналами. Такое описание свойственно языкам регистровых передач [4]. Организация процессов обеспечивается введением оператора процесса и оператора параллельного присваивания сигналов, также представляющего процессы. Это определяет две формы описания объектов на поведенческом уровне — потоковую и процессную.

Принцип повторного использования шаблонов СФЛМ является основополагающим в рассматриваемой системе распределённого проектирования. Т.е. структурно-функциональные модели, созданные одними проектировщиками, могут быть использованы другими для получения нового проектного решения, которое также сохраняется в базу знаний.

В качестве основы математического и программного обеспечения распределённой системы проектирования моделей СФЛМ предлагается использовать теорию многоагентных систем.

1. Структура распределённой мультиагентной системы

Агентно-ориентированные технологии представляют современный этап развития информационных систем. Мультиагентная система (МАС) представляет собой совокупность взаимосвязанных агентов, способных взаимодействовать друг с другом и окружающей средой, обладающих определёнными интеллектуальными способностями и возможностью индивидуальных и совместных действий [2].

В многоагентных системах инженерного проектирования множество агентов формируется из сущностей, подлежащих разработке. В качестве среды рассматривается разрабатываемый проект, который состоит из множества задействованных в проекте агентов. Задачи, порученные агентам, определяются инкапсулированными в них методами. Методы обеспечивают решение задач, порученных агентам, и выполнение операций, оказывающих воздействие на другие агенты [3].

Общая структура распределённой мультиагентной системы представлена в виде схемы, изображенной на рис. 1. Взаимодействие между серверами производится по технологии RPC (Remote Procedure Call), которая позволяет компьютерным программам вызывать функции или процедуры в другом адресном пространстве. К средствам коммуникации между агентами относятся языки FIPA-ACL (Foundation for Intelligent Physical Agents – Agent Communication Language) и KQML (Knowledge Query and Manipulation Language).

Основными компонентами архитектуры описываемого подхода являются агенты задачи, агенты ресурсов и окружающая среда. Окружающая среда содержит мобильные объекты и фиксированные объекты или ресурсы [5]. Мобильными объектами являются: проектное решение, проектная задача, шаблон структурно-функциональной модели, описание СФМ на языке VHDL.

Фиксированными объектами или ресурсами являются: ПК проектировщика, база знаний, общая рабочая память, блок постановки проектных задач, блок поиска, маршрутизатор, блок интеграции шаблона структурно-функциональной модели с графическим представлением СФМ, блок конструктора проектного решения, блок синтеза проектных решений.

Основу среды для рассматриваемого приложения представляет собой ориентированный граф, который изображён на рис.2., как часть общей системы. Узлами в графе являются

ресурсы в окружающей среде, ребра представляют связь между различными ресурсами. Мобильные объекты могут перемещаться от ресурса к ресурсу. Мобильные объекты могут перемещаться по графу и взаимодействовать с агентами на соответствующих ресурсах.

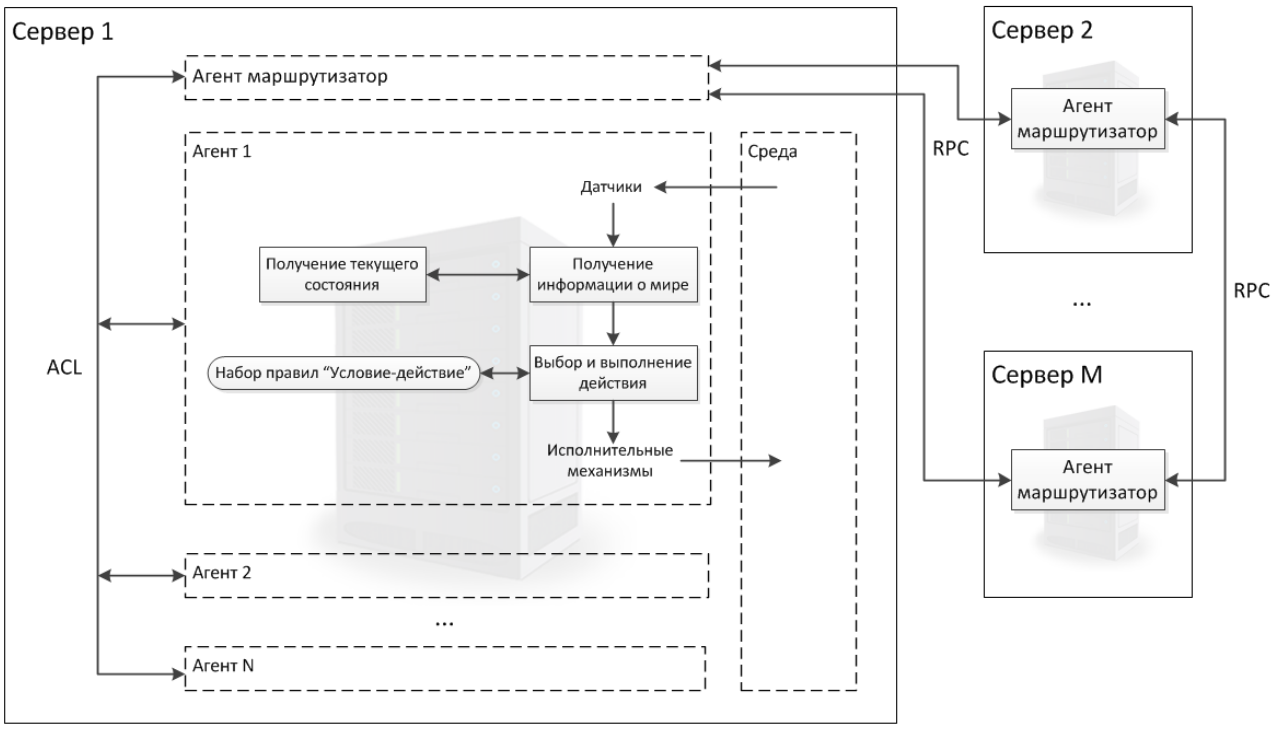

Рис. 1 – Общая структура распределённой мультиагентной системы

Объединения в команду проектировщиков, находящиеся на удалённом расстоянии друг от друга является важной задачей. Одним из путей решения этой задачи является создание web-интерфейса, с помощью которого проектировщики могут работать в одной системе, управляя проектными задачами и создавая проектные решения через браузер. Система, реализованная по облачной технологии SaaS (Software as a Service) позволяет использовать пользователям, которые не обладают доступом к серверу по локальной сети, функционал в виде услуг.

Распределенное проектирование с использованием технологии CORBA позволит эффективно распределять работу над проектом между несколькими проектировщиками, находящимися, как в одном месте, так и на удаленном расстоянии, при этом сохраняя контроль над проектом и системой [1]. Для коммуникации между серверами может использоваться IIOP CORBA или любой другой специфический протокол.

Общая структура мультиагентной системы распределённого проектирования структурно-функциональных моделей, представленных на языке VHDL, представлена на рис.2. Фиксированные объекты или ресурсы на схеме изображены в виде прямоугольников (например, "поисковик" представляет собой блок поиска шаблонов СФЛМ, проектных решений и проектных задач). Основу блока составляет агент, построенный с учётом BDI (beliefs-desires-intention) архитектуры. Интерфейсные агенты выполняют связующую роль мультиагентной системы с проектировщиками. Остальные блоки рассматриваемой системы построены по аналогичному принципу, за исключением функционального назначения. Блок постановки проектных задач необходим для создания и распределения проектных задач между проектировщиками. Блок конструктора проектных решений позволяет создавать проектное решение в виде шаблона СФЛМ на языке VHDL. Блок синтеза проектного решения позволяет объединять проектные решения, которые были созданы проектировщиками. Блок интеграции позволяет объединять шаблон СФЛМ с графическим представлением, которое описывает его работу. Маршрутизатор выполняет связующую роль

между блоками системы и другими серверами, которые исполняют роль хранения или разработки проектных решений.

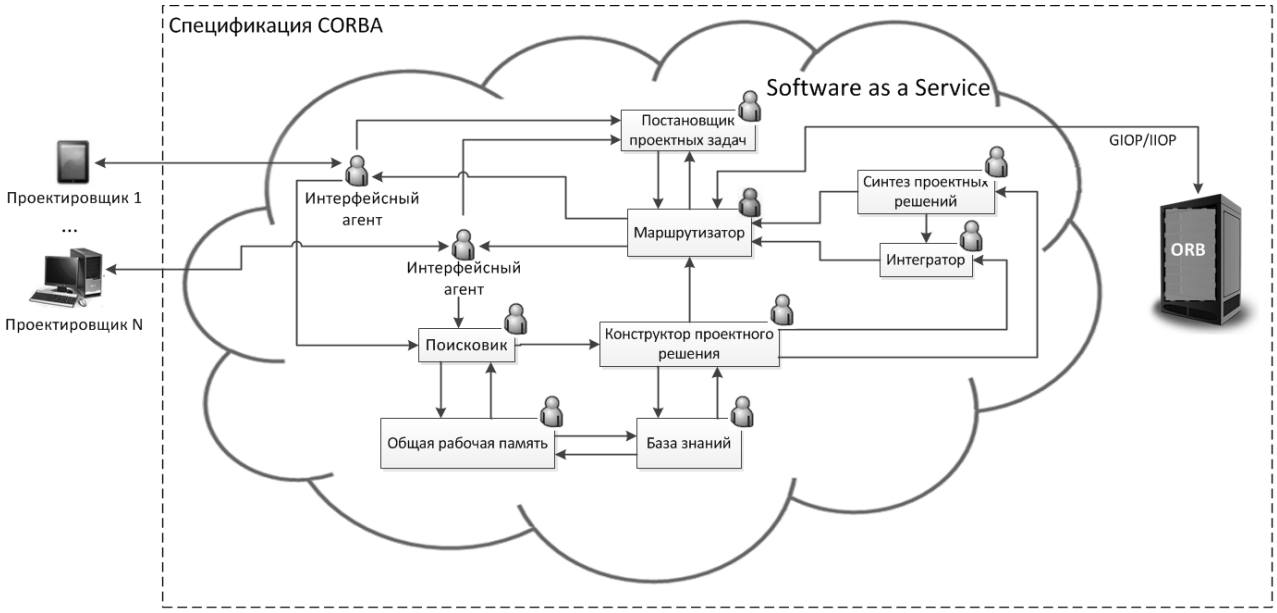

Рис. 2 – Общая структура мультиагентной системы распределённого проектирования структурно-функциональных моделей представленных на языке VHDL

Системы, использующие мультиагентную технологию для автоматизации проектирования, в основном направлены на решение задач, связанных с процессами расчета, поиска данных, оформлению результатов, решения сложных задач управления ресурсами и управления производством изделий, от поступления заказа до выпуска продукции. Агенты обладают возможностью синтеза структуры представляемых ими объектов и подбора параметров элементов, их составляющих. Также агент может получить информацию о характеристиках объектов в виде макромоделей. В некоторых системах есть поддержка работы групп разработчиков, которые могут находиться в различных частях света.

Ввиду этого остается актуальным применение мультиагентной технологии при работе с СФЛМ, использующими языки описания, такие как VHDL. Инструментальные средства должны обеспечивать генерацию текстового представления баз знаний на языке, который будет понятен проектировщику, и генерацию эффективных программных кодов, для улучшения которых не нужно привлекать программистов. Новый подход должен быть ориентирован на иной способ использования интеллектуальной многоагентной системы для работы с СФЛМ. Решение исходной задачи проектирования должно быть получено в результате процессов кооперации и конкуренции внутри многоагентной системы, т.е. роль проектировщика, в основном, сводится к постановке задачи и формированию программного кода на языке VHDL, если необходимое решение не было найдено в базе знаний.

Выводы

Следует отметить, что поскольку в процесс проектирования сложных системы вовлечено множество проектировщиков, обладающих разнообразными знаниями в различных областях, то важной задачей автоматизации данного процесса является решение задач построения единой информационной системы, использование которой способствует увеличению производительности как отдельного проектировщика в частности, так и команды в целом. Успех применения МАС в САПР во многом определяется качеством проектирования инструментальных средств поддержки МАС в САПР. Также мультиагентный подход можно использовать в тех случаях, когда приоритетным является экономия рабочего времени проектировщика за счет передачи части рутинных операций, например, поиска проектных решений или шаблонов СФЛМ.

Литература

1. Хородов В.С., Игонин А.Г. Технологии распределённого проектирования. // Вестник УлГТУ, 2014. – №1. – С. 55-59.
2. Мутовкина Н.Ю., Кузнецов В.Н., Клюшин А.Ю., Палюх Б.В. Нечёткие методы согласованного управления в многоагентных системах // Вестник ТГТУ, Том 19. – №4.
3. Тарасов В.Б. От многоагентных систем к интеллектуальным организациям: Философия, психология, информатика. – М.: УРСС, 2002.
4. Ивченко В.Г. Применение языка VHDL при проектировании специальных СБИС. Т.: ТГРУ, 2000. – С. 1-15, 45-47.
5. Holvoet T., Valckenaers P. Exploiting the Environment for Coordinating Agent Intentions. // Environments for Multi-Agent Systems III. Lecture Notes in Computer Science, 2007. pp. 51-66.

УДК 004.891

КЛАССИФИКАЦИЯ УПРАВЛЯЮЩИХ ВОЗДЕЙСТВИЙ В СИСТЕМАХ ЭНЕРГЕТИЧЕСКОГО МЕНЕДЖМЕНТА ОРГАНИЗАЦИИ, НА ОСНОВЕ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПОДХОДА

Болдырев Владислав Вячеславович, студент, Комсомольский-на-Амуре государственный технический университет, Россия, Комсомольск-на-Амуре, boldurev16@gmail.com
Горькавый Михаил Александрович, канд. техн., наук, доцент, Комсомольский-на-Амуре государственный технический университет, Россия, Комсомольск-на-Амуре, idpo@knastu.ru

Введение

Энергоэффективность и энергосбережение – это одно из приоритетных направлений развития и науки РФ. Большинство решений в этой области связанно с созданием программно-технических комплексов, предназначенных для управления энергией с целью поддержания её бережливого потребления.

Любая подобная разработка является сложной системой, включающей множество взаимосвязанных процессов. В настоящее время основными механизмами данных систем являются люди и программно-технические средства, где люди преимущественно выступают в качестве субъектов управления. Но при этом большинство отклонений в работе этих систем возникает по причине наличия у субъекта управления такой особенности, как человеческий фактор.

В предыдущих работах авторов [1],[2] было предложено разработать интеллектуальную систему энергосбережения, особенностью, которой является минимизация негативного воздействия человеческого фактора на управление энергетическими ресурсами. Однако эффективность работы такой системы будет недостаточной, если не обеспечить взаимодействие между программно-техническими средствами и ЛПР.

В настоящей статье представлена диаграмма классов интеллектуальной системы энергосбережения и методы воздействия на субъекты управления с применением программно-технических средств системы.

1. Атрибуты и диаграмма классов субъектов управления

Прежде чем определить методы воздействия, необходимо сформировать набор атрибутов субъекта управления, которые позволят ему адекватно реагировать на поступающие воздействие от системы. В таблице 1 приведены атрибуты субъектов управления.

Интеллектуальная система энергосбережения предполагает минимизацию человеческого фактора не путем уменьшения присутствия ЛПР в управлении, а путем воздействия системы на него. Т.е. субъектом управления может являться как одно ЛПР, так и

несколько, без увеличения риска возникновения ошибок, связанных с человеческим фактором; напротив, коллективное решение с большей вероятностью будет эффективным.

Атрибуты, приведенные в таблице, будут учитываться системой при формировании рекомендаций для субъектов управления, но для того, чтобы это реализовать, необходимо классифицировать субъекты управления и присвоить к каждому из классов определенный вид воздействий.

Таблица 1. Атрибуты субъектов управления

Субъект управления	**Способ принятия решения**	**Атрибуты**
ЛПР	Единолично	Набор компетенций
		Вид деятельности
		Состояние здоровья (уровень зрения)
		Набор знаний
	Коллективно	Численность
		Групповые нормы
		Групповые санкции

На рисунке 1 представлена диаграмма классов субъектов управления системой энергосбережения аудитории университета, в которой проводятся лекции, практические занятия с использованием ПК, лабораторные работы, защиты с использованием мультимедийного оборудования, а также осуществляется уборка помещения.

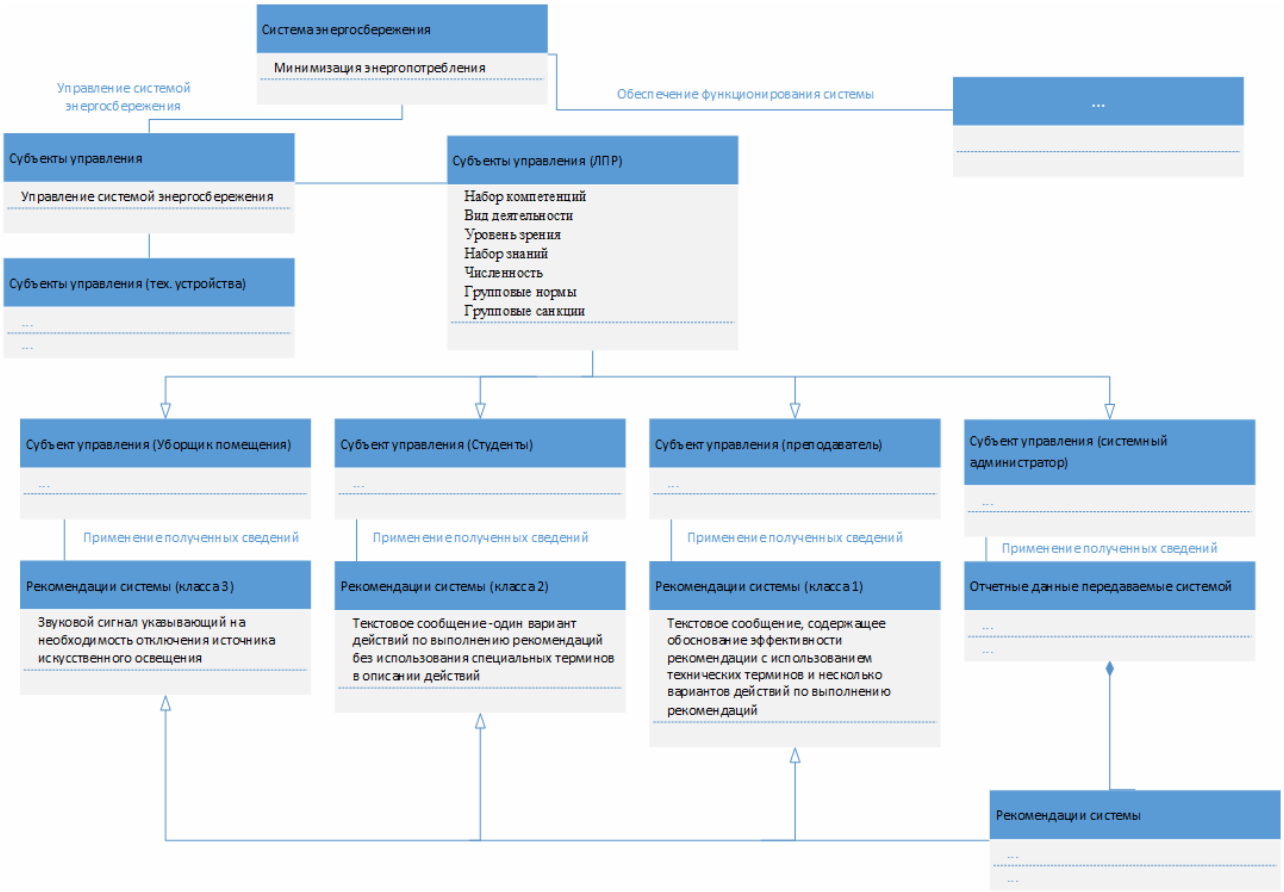

Рис. 1 – Диаграмма классов

При создании рисунка использовалась общепринятая система обозначений (UML стандарт).

На рисунке отображены основные классы субъектов управления системы энергосбережения:

Системный администратор – осуществляет мониторинг работы системы и задает базовые параметры, к которым относятся требования к освещенности рабочих мест;

Преподаватель – оперативно регулирует мощность искусственных источников света в случае поступления сообщения от системы;

Студент (студенты) – оперативно регулирует мощность искусственных источников света в случае поступления сообщения от системы и\или требования преподавателя;

Уборщик помещения – оперативно регулирует мощность искусственных источников света в случае появления звукового сигнала.

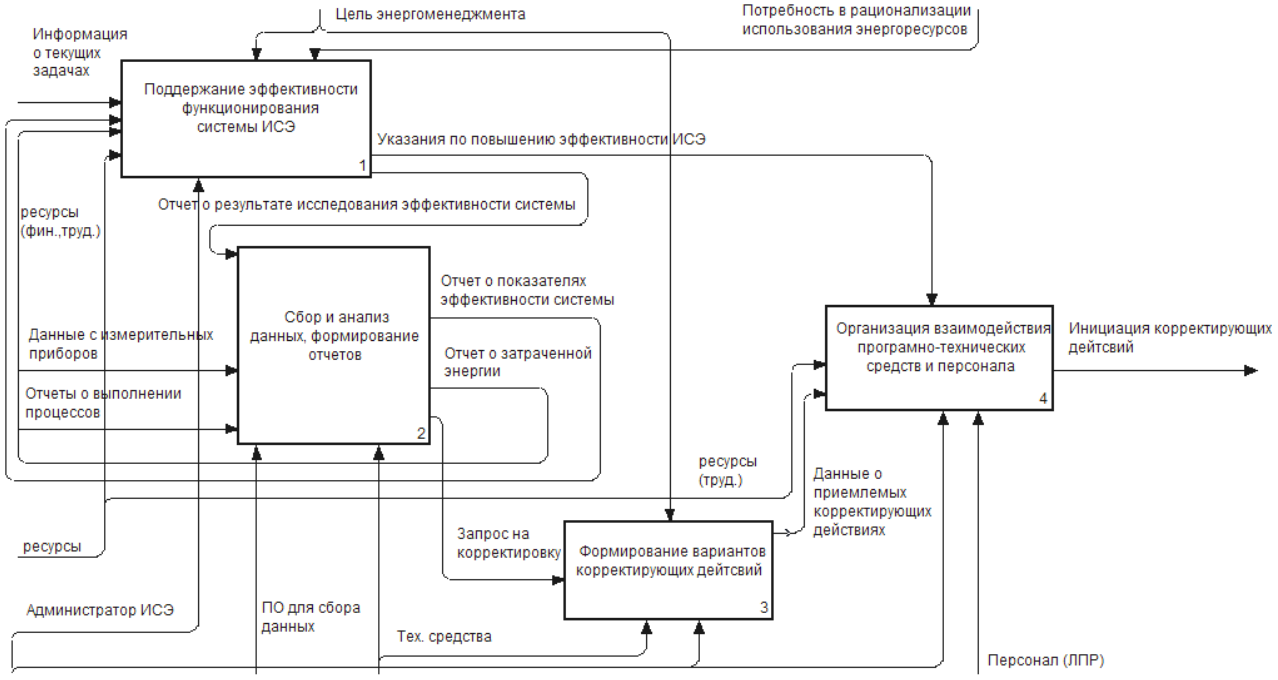

Рис. 2 – верхний уровень процессной модели функционирования “ИСЭ”

Диаграмма выше отображает структуру управляющих воздействий на ЛПР, участвующих в осуществлении корректирующих действий, необходимых для эффективного функционирования системы энергоменеджмента. Далее на рисунке 2 приведен верхний уровень процессной модели функционирования интеллектуальной системы энергоменеджмента (система обозначений соответствует стандарту IDF0).

Представленная диаграмма не является универсальной, но отображает принцип взаимодействия ЛПР и программно- технического комплекса, через генерирование комплексом сообщений различного содержания и их передачу в рамках процесса «Организация взаимодействия программно-технических средств и персонала».

Информация, которую необходимо донести до субъекта управления для эффективного функционирования системы, подается в виде текстового сообщения или звукового сигнала. Звуковой сигнал является наименее информативным сообщением и предназначен для одной цели – указать на необходимость отключения искусственного источника питания. Данный вид сигнала наиболее прост в понимании и поэтому направлен преимущественно на субъекты управления с наименьшим количеством атрибутов. В представленном примере уборщик помещения имеет наименьшее количество атрибутов, поскольку уборка относится к категории работ Iа и имеет наименьшее количество требований к освещенности рабочей поверхности. Кроме того, понимание звукового сигнала не требует специальных компетенций и знаний и поэтому может использоваться в качестве воздействия на наиболее неквалифицированный персонал.

Рекомендации в виде тестовых сообщений, в свою очередь, могут быть двух видов:

1. Текстовое сообщение с планом действий по выполнению рекомендаций без использования специальных терминов, изложенное на максимально доступном естественном языке, – позволяет оперативно устранить причину отклонения энергопотребления от нормы

ЛПР или группе ЛПР, имеющих общие технические знания.

В примере к данному классу субъектов управления относятся студенты технического университета, обладающие набором знаний, позволяющих выполнить рекомендации.

2. Текстовое сообщение с обоснованием эффективности предложенных вариантов устранения отклонения, построенное на использовании специальных технических терминов, – позволяет выявить и устранить причину возникновения отклонений.

В примере к данному классу относится преподаватель – ЛПР, обладающее набором знаний и компетенций, позволяющих ему адекватно воспринять полученную от системы информацию и исправить ошибки в энергопотреблении.

Данные, полученные от измерительных приборов и обработанные системой, необходимы системному администратору для поддержания эффективности работы системы. Предоставляются оператору в виде развернутых отчетов.

Сообщения, генерируемые системой для каждого из классов, формируются с учетом их атрибутов. Наиболее важными являются набор знаний и компетенций, но также важно учитывать состояние здоровья субъектов управления, т.к. без учета этого атрибута возможно снижение его работоспособности.

Заключение

Унифицированная диаграмма классов для систем энергетического менеджмента позволяет установить виды взаимодействия между программно–техническим обеспечением и ЛПР. Где ЛПР – это субъект управления, т.е. часть системы энергетического менеджмента. Поэтому эффективное функционирование данной системы возможно только при наличии обратной связи от объекта (программно-техническое обеспечение) к субъекту управления (ЛПР) в виде названых выше сигналов.

Литература

1. Иванов С.А., Разработка интеллектуальной системы энергоменеджмента на основе объектно-ориентированного подхода/ С.А.Иванов, Л.А. Вяль, М.А. Горькавый// Объектные системы 2013: Материалы VII Международной научно-практической конференции, Россия, Ростов-на-Дону 10-12 мая 2013 г., С. 45-50.
2. Болдырев, В.В. Концепция интеллектуального алгоритма автоматизированной системы энергопотребления/ В.В. Болдырев, М.А. Горькавый // Технические и математические науки: актуальные проблемы и перспективы развития: материалы междунар. науч.-практ. конф., Саратов, 14.11.2013г., С. 19 – 24.
3. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений на С++ / Г. Буч; пер. с англ. И. Романовского; под ред. Ф. Андреева – М.: Невский Диалект, 2000. – 359 с.

УДК 004.4'236

ФУНКЦИОНАЛЬНАЯ СХЕМА ВСТРОЕННОГО ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ИНТЕРАКТИВНОГО МЕХАНИЗМА КОНСТРУИРОВАНИЯ СТРАНИЦ САЙТА НА ОСНОВЕ МАТРИЧНОЙ УНИВЕРСАЛЬНОЙ ОБЪЕКТНО-РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ

Микляев Иван Александрович, к.ф.-м. н., доцент, Институт судостроения и морской арктической техники (Севмашвтуз) Северного (Арктического) федерального университета имени М. В. Ломоносова, Россия, Северодвинск, ivanmia1@rambler.ru

Юрецкий Алексей Павлович, студент, Институт судостроения и морской арктической техники (Севмашвтуз) Северного (Арктического) федерального университета имени М. В. Ломоносова, Россия, Северодвинск, admin@webprogrammist.su

Введение

При разработке сайтов для ускорения и формализации процесса используются различные виды программ:

1. Среды визуальной разработки, например, Dreamweaver [1], FrontPage [2].
2. Интегрированные среды разработки (IDE), например Zend Studio [3], Delphi for PHP [4].
3. Среды веб-разработки – системы управления сайтом (CMS), например 1C-Bitrix [5], Wordpress [6].

Разработка встроенного объектно-ориентированного интерактивного механизма конструирования страниц сайта (ВОИМКСС) на основе матричной универсальной объектно-реляционной базы данных (МУОРБД) [7] преследует цель создание интегрированной веб-среды визуальной разработки сайтов, как доступной пользователям, не имеющим специальных знаний в сфере веб-разработки, так и подходящей в качестве мощного инструмента для профессиональных разработчиков, обеспечивающей гибкость в возможностях настройки дизайна, не ограниченную использованием стандартных шаблонов, обеспечивающую широкий спектр функционала, доступного для расширения непосредственно разработчиком, лёгкость развёртывания на хостинге и простоту переноса сайта.

1. Прикладное применение МУОРБД для размещения структуры сайтов

Формирование древовидного представления информации, содержащейся в базе данных, позволяет проводить любые динамические изменения и расширения структуры.

МУОРБД развёрнута на платформе MySQL. Все действия с информацией структуры сайта, размещённой в МУОРБД, позволяет задействовать весь механизм как самой МУОРБД, так и инструментарий реляционной платформы [8].

Очевидным является в условиях реальной разработки использование интерактивной среды, что ещё в 1990-е годы показало значительное преимущество. Оптимальным решением для быстрой и наглядной разработки является способ редактирования, который получил название WYSIWYG (от англ. What You See Is What You Get — «что видишь, то и получишь»), при котором корректируемый материал в процессе редактирования выглядит в точности так же, как и конечный результат. Для осуществления этого способа редактирования МУОРБД-сайтов разработан ВОИМКСС.

Визуально ВОИМКСС представляет собой комплекс интерактивных элементов, расположенных в слое над слоем страницы. ВОИМКСС, в отличие от ныне существующих систем, имеет прямую связь с древовидной структурой информации, которая размещается в МУОРБД. МУОРБД-ключ, который присваивается каждому элементу страницы, показывает адрес данного элемента в древовидной структуре.

2. Организация ВОИМКСС

ВОИМКСС имеет сложную организацию с множеством логических ветвлений. Для упрощения работы и понимания происходящих процессов была разработана структура переменных и функций. Она позволяет в любой точке исполняемого кода иметь либо получать необходимые данные о состоянии системы. Тем самым оптимизируется процесс модернизации, расширения и верификации исполняемого кода ВОИМКСС и его дальнейшая работа на уровне пользователя.

ВОИМКСС функционирует на стороне клиента. Обмен данными с сервером осуществляется через технологию межъязыкового взаимодействия Ajax.

3. Описание механизма взаимодействия

После того, как пользователь авторизовался на странице для её редактирования, страница перегружается в окне браузера, неся с собой функционал ВОИМКСС. (см. рис. 1 – начальное псевдосостояние «Начало»).

В состав функционала из справочной сущности МУОРБД генерируются заданные массивы возможных действий и состояний системы. Сразу после того, как загрузится скрипт, активируются функции перехвата событий и система находится в состоянии ожидания наступления события (см. рис. 1 – сущность «Ожидание наступления события», переход «Передача сигнала об отсутствии события»). Теперь все действия пользователя в браузере контролируются программой.

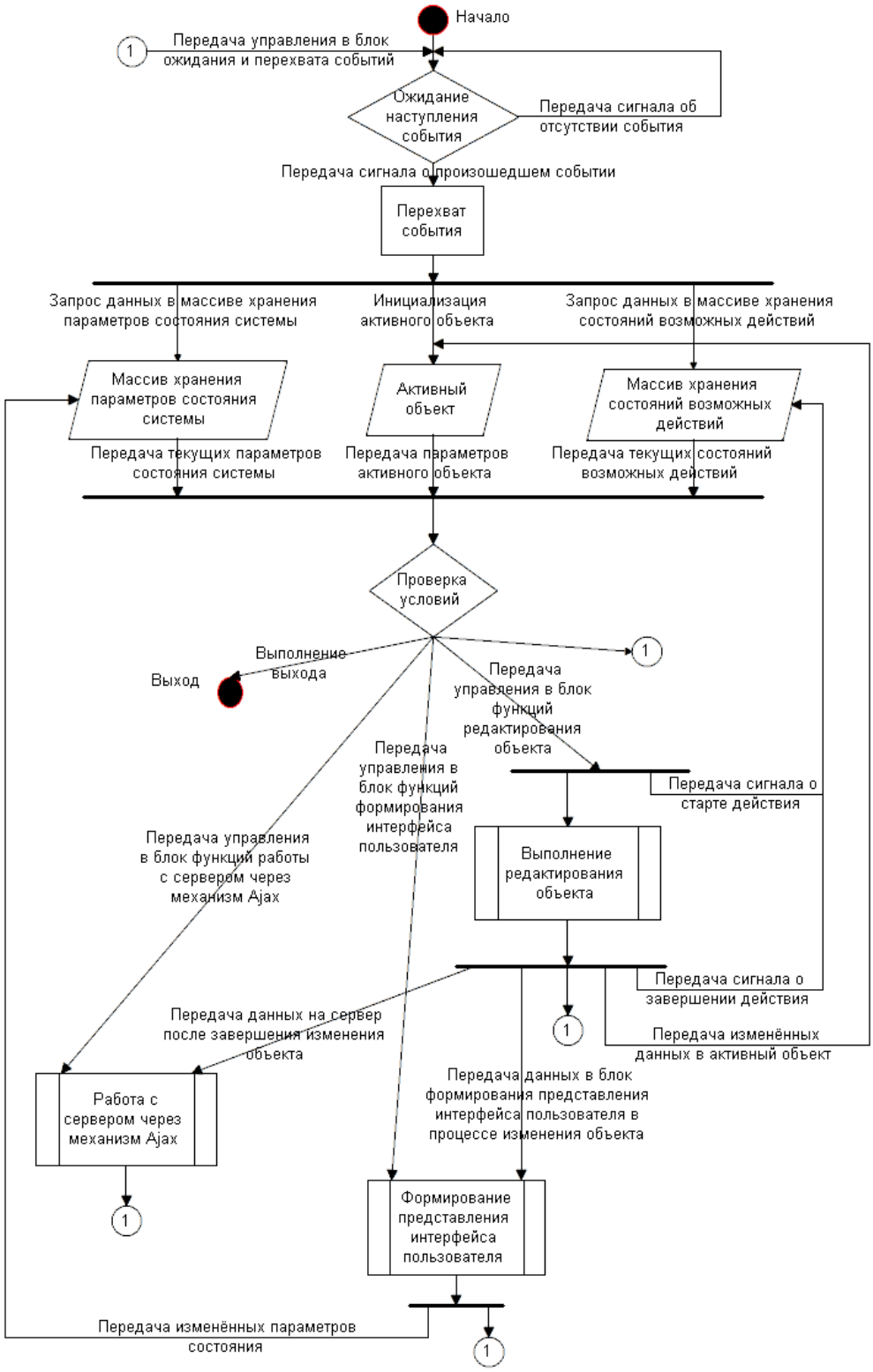

Рис. 1 – Схема взаимодействия структурных элементов ВОИМКСС

Когда пользователь выполняет действие на странице браузера (см. рис. 1 – переход

«Передача сигнала о произошедшем событии»), начинает работу соответствующая функция перехвата событий (см. рис. 1 – сущность «Перехват события»). В первую очередь извлекается информация о том, на каком элементе произошло событие, создаётся клон элемента объектной модели браузера и заносится его в глобальную хеш-переменную «Активный объект» (см. рис. 1 – переход «Инициализация активного объекта», сущность «Активный объект»), а также запрашиваются данные в массивах хранения параметров состояния системы и состояний возможных действий (см. рис. 1 – переходы «Запрос данных в массиве хранения параметров состояния системы», «Запрос данных в массиве хранения состояний возможных действий», сущности «Массив хранения параметров состояния системы», «Массив хранения состояний возможных действий»).

Отдельным случаем является событие «onclick» на главных управляющих элементах – кнопках «Выход» и «Просмотр / Редактирование». Обработка этих событий осуществляется вне основной схемы. В случае нажатия на кнопку «Выход» осуществляется завершение работы ВОИМКСС и перегрузка страницы без его функционала. Кроме того, в серверной части программы запускается система сохранения изменений страницы сайта и происходит уничтожение сессии, которая хранит авторизацию пользователя на текущей странице для её редактирования (см. рис. 1 – сущность «Проверка условий», переход «Выполнение выхода», конечное псевдосостояние «Выход»).

При первоначальной загрузке функционала ВОИМКСС переменная-флаг, хранящая метку о глобальном состоянии системы, получает значение «False», что соответствует состоянию просмотра страницы. Нажатие кнопки «Просмотр / Редактирование» меняет значение этого флага на противоположное. После чего происходит формирование соответствующего флагу представления интерфейса пользователя (см. рис. 1 – переход «Получение параметров активного объекта», сущность «Проверка условий», переход «Передача управления в блок ожидания и перехвата событий»), в котором изменяется подпись кнопки «Просмотр / Редактирование», отображаются панель компонентов и панель атрибутов. Далее система вновь находится в состоянии ожидания наступления события.

В остальных случаях, исходя из текущих параметров состояния системы, состояния возможных действий и данных об активном объекте (см. рис. 1 – сущности «Массив хранения параметров состояния системы», «Массив хранения состояний возможных действий», «Активный объект», переходы «Передача текущих параметров состояния системы», «Передача текущих состояний возможных действий», «Передача параметров активного объекта»), управление передаётся в соответствующий блок функций, где в дальнейшем протекает предопределённый процесс (см. рис. 1 – переходы «Передача управления в блок функций редактирования объекта», «Передача управления в блок функций формирования интерфейса пользователя», «Передача управления в блок функций работы с сервером через механизм Ajax»).

Общая деятельность системы не зависит от внутреннего содержания любого из блоков функций ВОИМКСС, а определяется лишь выбором из них, передаваемой и получаемой информацией. Таким образом, блоки функций могут расширяться неограниченно. Структура передаваемой и получаемой информации одинакова для всех вызываемых функций одного блока; для функций различных блоков отличается.

Передача управления функции из блока формирования представления интерфейса пользователя – первый вариант хода логической цепочки (см. рис. 1 – переход «Передача управления в блок функций формирования интерфейса пользователя», сущность «Формирование представления интерфейса пользователя») частично был описан выше, когда речь шла о формировании соответствующего флагу интерфейса пользователя.

На выходе из предопределённого процесса формирования представления интерфейса пользователя передаются данные об изменённых параметрах состояния системы, которые фиксируются в массиве хранения параметров состояния системы, и передаётся управление в блок ожидания и перехвата события (см. рис. 1 – переходы «Передача изменённых параметров состояния», «Передача управления в блок ожидания и перехвата событий»).

В данном случае изменения не затронут редактируемую страницу. Данные на сервере также не будут изменены. Изменится только внешнее отображения интерфейса ВОИМКСС.

Если кликнуть на какой-либо элемент страницы, происходит событие «onclick», в результате которого клон объекта помещается в локальную переменную «Активный объект». Здесь исключена обработка событий на кнопках «Выход» и «Просмотр / Редактирование», поэтому работа механизма идёт по основному направлению логической цепочки схемы. Система запущена на редактирование, поэтому далее идёт проверка по условиям. Далее происходит вызов функции, соответствующей данным условиям, которая относится к блоку формирования представления интерфейса пользователя.

В результате работы функции происходит изменение интерфейса пользователя, а именно: скрывается панель компонентов; происходит выделение активного элемента; появляется панель с атрибутами и параметрами выбранного элемента (Object Inspector).

После выполнения функции из блока формирования представления интерфейса пользователя происходят соответствующие изменения в массиве хранения параметров состояния системы. Панель компонентов помечается как скрытая; Object Inspector помечается как отображаемый.

Далее система опять возвращается в состояние ожидания события.

Передача управления функции из блока функций редактирования объектов – второй вариант хода логической цепочки, основой (см. рис. 1 – переход «Передача управления в блок функций редактирования объекта», сущность «Выполнение редактирования объекта»). Перед началом выполнения предопределённого процесса редактирования объекта вносятся изменения в массив хранения возможных состояний системы – фиксируется старт действия, которое запускается вызовом функции, выбранной в соответствии с текущими условиями (см. рис. 1 – переход «Передача сигнала о старте действия»). На выходе из предопределённого процесса редактирования объекта происходит несколько действий:

1. Изменяется активный объект (см. рис. 1 – переход «Передача изменённых данных в активный объект»).
2. Происходит передача управления в предопределённый процесс формирования представления интерфейса пользователя (см. рис. 1 – переход «Передача данных в блок формирования представления интерфейса пользователя в процессе изменения объекта»).
3. Происходит передача данных в предопределённый процесс работы с сервером через механизм Ajax (см. рис. 1 – переход «Передача данных на сервер после завершения изменения объекта»).

После завершения предопределённого процесса редактирования объекта, вносятся изменения в систему хранения возможных состояний системы (см. рис. 1 – переход «Передача сигнала о завершении действия») – фиксируется завершение действия, которое было запущено; даётся команда на завершение действия, которая возвращает систему в состояние ожидания события (см. рис. 1 – переход «Передача управления в блок ожидания и перехвата событий»). При движении по этой логической цепочке схемы, происходят изменения, которые в зависимости от вызванной функции могут коснуться редактируемой страницы, данных на сервере или представления интерфейса пользователя.

В качестве примера рассмотрим выполнение действия перемещения элемента страницы.

Перед началом перемещения элемента происходит событие «onmouseover». Клон объекта помещается в локальную переменную «Активный объект». Это событие не «onclick», где необходимо исключать обработку основных функциональных кнопок «Выход» и «Просмотр / Редактирование», значит, работа программы идёт по общему случаю. Система запущена на редактирование, что по логической цепочке схемы ведёт к проверке по условиям. Происходит вызов функции перемещения. Эта функция относится к блоку редактирования объекта.

Далее фиксируется старт возможного активного действия «перемещение» в системе хранения возможных действий, а также сохраняется идентификатор перемещаемого объекта.

В процессе перемещения происходит постоянная передача информации в функцию блока формирования представления интерфейса пользователя, которая в режиме реального времени фиксирует изменение параметров перемещаемого объекта (отступ сверху и отступ слева) и отображает эти изменения в Object Inspector.

После завершения перемещения, когда пользователь отпускает кнопку мыши, завершается работа функции перемещения. Одновременно с этим фиксируется завершение активного действия «перемещение» в системе хранения возможных действий и высвобождается идентификатор перемещаемого объекта, вызывается функция из блока работы с сервером через механизм Ajax, которая преобразует параметры перемещённого элемента в необходимый вид и передаёт их на сервер посредством технологии Ajax. Сервер, получив данные, обрабатывает их определённым образом и обновляет изменённые значения в сессии.

Далее система опять возвращается в состояние ожидания события.

Передача управления функции из блока функций работы с сервером через механизм Ajax – третий вариант хода логической цепочки (см. рис. 1 – переход «Передача управления в блок функций работы с сервером через механизм Ajax», сущность «Работа с сервером через механизм Ajax»). На входе в предопределённый процесс используются данные активного объекта, которые преобразуются в необходимый вид и передаются на сервер посредством технологии Ajax. На выходе происходит возврат системы в состояние ожидания события (см. рис. 1 – переход «Передача управления в блок ожидания и перехвата событий»). В данном случае изменений на редактируемой странице и в представлении интерфейса пользователя не происходит. Изменяются данные на сервере.

Например, произведя необходимые изменения на странице сайта и убедившись, что получили то, что необходимо, можно зафиксировать изменения, перенеся их из сессии в базу данных. Для этого, служит кнопка «Сохранить изменения». Происходит событие «onclick». Здесь исключена обработка событий на кнопках «Выход» и «Просмотр / Редактирование», поэтому работа механизма идёт по основному направлению логической цепочки схемы. Система запущена на редактирование, поэтому далее идёт проверка по условиям.

Вызывается функция из блока работы с сервером через механизм Ajax, которая передаёт на сервер сигнал о том, что необходимо зафиксировать произведённые изменения. Сервер, получив этот сигнал, выполняет функцию, которая формирует запросы, в которых заложены внесённые изменения и отправляет их на сервер баз данных.

Далее система опять возвращается в состояние ожидания события.

4. Структурные элементы

Структура ВОИМКСС состоит из двух двухмерных массивов, переменной-хеша и четырёх групп функций, которые объединены в соответствии с выполняемыми действиями.

Таким образом, в ВОИМКСС используются следующие структурные элементы:

- Массив хранения состояний возможных действий – двухмерный массив, содержащий состояния всех возможных (реализованных) на данный момент действий в системе и их параметры (см. рис. 1 – сущность «Массив хранения состояний возможных действий»).
- Массив хранения параметров состояния системы – двухмерный массив, содержащий все необходимые для отслеживания состояния системы в целом, либо отдельных её элементов (см. рис. 1 – сущность «Массив хранения параметров состояния системы»).
- Активный объект – ассоциативный javascript-массив, хеш, содержащий соответствия «ключ» => «значение», и методы, соответствующие реальному объекту, с которым в текущий момент времени производит действие пользователь (см. рис. 1 – сущность «Активный объект»). Функции трансляции-коммуникации. Это группа функций, которые посредством технологии Ajax отправляют на сервер информацию о

результатах работы системы и получают необходимые для работы дополнительные данные (см. рис. 1 – сущность «Работа с сервером через механизм Ajax»).

- Функции элементов управления визуальным отображением интерфейса. Это группа функций, которые отвечают за работу элементов управления визуальным отображением ВОИМКСС, то есть обеспечивают интерфейс пользователя (см. рис. 1 – сущность «Формирование представления интерфейса пользователя»).
- Функции редактирования. Это группа функций, которые реализуют возможности интерактивной работы пользователя с элементами редактируемой страницы (см. рис. 1 – сущность «Выполнение редактирования объекта»).
- Функции перехвата событий. Это центральная группа функций, которые запускают ожидание действий пользователя и определяют алгоритм работы программы в зависимости от того, какое действие произвёл пользователь, с каким элементом системы и в каком состоянии в текущий момент времени находятся значимые в каждом конкретном случае элементы системы (см. рис. 1 – сущности и переходы между начальным псевдосостоянием «Начало» и переходами после сущности «Проверка условий»).

Заключение

ВОИМКСС позволяет:

- работать с содержимым редактируемой страницы «по месту»,
- создавать html-страницы;
- добавлять и удалять элементы на html-страницах;
- перемещать элементы при помощи мыши;
- изменять размер элементов при помощи мыши;
- изменять значения атрибутов и параметров стилей элементов через Object Inspector;
- работать с текстовым содержимым элементов;
- сохранять внесённые изменения в сессию и в базу данных.

Методика разработки механизма учитывает возможность расширения функционала как в ширину, за счёт добавления любого количества функций в блоках (перехвата событий, формирования представления интерфейса пользователя, редактирования объекта, работы с сервером через механизм Ajax) без их реконструкции, а также в глубину, за счёт возможности увеличения размерности массива хранения состояний возможных действий, что позволяет допустить одновременную активность двух действий одного ранга или даже одновременное выполнение двух действий над различными объектами. В результате появляется возможность оперировать ВОИМКСС неограниченным количеством активных объектов.

Литература

1. Dreamweaver CC, http://www.adobe.com/ru/products/dreamweaver.html
2. Знакомство с Microsoft Office FrontPage 2003, http://office.microsoft.com/ru-ru/frontpage-help/HA001071499.aspx?CTT=1
3. Zend Studio – The Leading PHP IDE, http://www.zend.com/products/studio/
4. Delphi for PHP 2: прежним курсом, http://ko.com.ua/delphi_for_php_2_prezhnim_kursom_37552
5. 1С-Битрикс – система управления сайтом, http://www.1c-bitrix.ru/
6. Wordpress, http://ru.wordpress.org/
7. Микляев И.А., Матричная универсальная объектно-реляционная база данных// Материалы, 1 Международная научно-практическая конференция Объектные Системы – 2010, Ростов-на-Дону, 2010, стр. 34-39
8. Микляев И.А. Матричная универсальная объектно-реляционная база данных на реляционной платформе: монография/ Сев.(Арктич.) федер. Ун-т им. М.В. Ломоносова. – Архангельск: ИД САФУ, 2014. – 226 с.

УДК 004.4, 004.65

ИНТЕГРАЦИЯ SQL-ОРИЕНТИРОВАННЫХ СУБД И NOSQL-СИСТЕМ НА УРОВНЕ ОБЪЕКТНОГО ОТОБРАЖЕНИЯ[1]

Посконин Андрей Владимирович, аспирант кафедры Системного программирования факультета Вычислительной математики и кибернетики Московского государственного университета им. М. В. Ломоносова, Россия, Москва, aposk@yandex.ru

Современные приложения вынуждены работать с данными различной структуры, объема и степени важности, предъявляя к системам хранения данных достаточно жесткие (а порой и невыполнимые) требования. Отсутствие универсального решения для работы с данными привело к появлению на рынке большого числа узкоспециализированных систем, значительно отличающихся от традиционных SQL-ориентированных СУБД. Этот новый класс систем получил собирательное название «NoSQL» («Not Only SQL»), подчеркивая тем самым, что в ряде задач применение реляционных СУБД может быть не самым эффективным решением.

Стремительное развитие Интернет и Web-приложений вывело на первый план проблему масштабируемости при возрастающих нагрузках и объемах информации, которую NoSQL-системы стараются решить с использованием распределенных архитектур в противоположность классическим «односерверным» СУБД. Однако поддержка согласованности данных (и, особенно, поддержка ACID-транзакций) в условиях распределенности увеличивает не только сложность системы, но и время отклика, из-за большого числа сетевых взаимодействий. По этой причине NoSQL-системы ослабляют гарантии согласованности данных и отказываются от ACID-транзакций в пользу масштабируемости, скорости и высокой доступности данных (см., например, [1]). Системы категории NoSQL, кроме того, основаны на нереляционных моделях данных (например, ключ-значение, документная модель и т.д.), ориентированы в основном на работу с неструктурированными или слабоструктурированными данными и используют отличные от SQL языки запросов и интерфейсы доступа к данным. Стоит также упомянуть о появлении нового поколения SQL-ориентированных СУБД, изначально создаваемых для работы в распределённой среде, – так называемых NewSQL-системах. Эти СУБД поддерживают транзакционную семантику и язык SQL, но обеспечивают при этом приемлемую масштабируемость. Подробные обзоры существующих NoSQL- и NewSQL-решений можно найти, например, в [2-4].

Рис. 1 – Реляционная и документная модели данных

[1] Статья рекомендована к опубликованию в журнале "**Информационные технологии**"

Богатый выбор различных систем для управления данными обусловлен тенденцией к специализации инструментов: если раньше SQL-ориентированные системы рассматривались как универсальные СУБД, способные решать практически любые задачи, то сегодня для каждой задачи можно выбрать наиболее оптимальное и удобное решение (см., например, [5]). Например, если приложению приходится работать с сущностями, обладающими переменным набором атрибутов, то практичнее использовать NoSQL-системы с документной моделью данных, чем реализовывать модель Entity-Attribute-Value [6] средствами SQL. Документная модель данных позволяет хранить объекты с произвольным набором атрибутов, обычно представляемые в JSON и называемые «документами». При этом обычно допускаются списки и вложенные документы, поддерживаются индексы и выборки на основе полей документов. Сами документы обычно хранятся в коллекциях, не накладывающих каких-либо ограничений на набор атрибутов входящих в них документов (отсутствие строгой схемы данных). Одной из наиболее популярных систем такого типа является MongoDB. Эта система имеет богатую функциональность и поддерживает атомарные операции на уровне одного документа, однако если приложение нуждается в полноценных ACID-транзакциях, то SQL-ориентированные системы по-прежнему вне конкуренции.

Термин «Polyglot Persistence» [7] применяется в англоязычных источниках для обозначения практики использования нескольких различных систем хранения данных в рамках одного приложения. Потребность в этом обусловлена тем фактом, что современные приложения (а особенно Web-приложения) вынуждены работать одновременно с несколькими видами данных.

Например, это может быть информация о клиентах, каталог товаров, журналы событий и переходов по ссылкам, данные пользовательских сессий и т.д. Все эти данные не только обладают различной структурой и соотношением операций чтения/записи, но и предъявляют разные требования к надежности хранения и возможностям языка запросов. Таким образом, естественно применить для работы с этими данными несколько систем хранения (см., например, [8-9]):

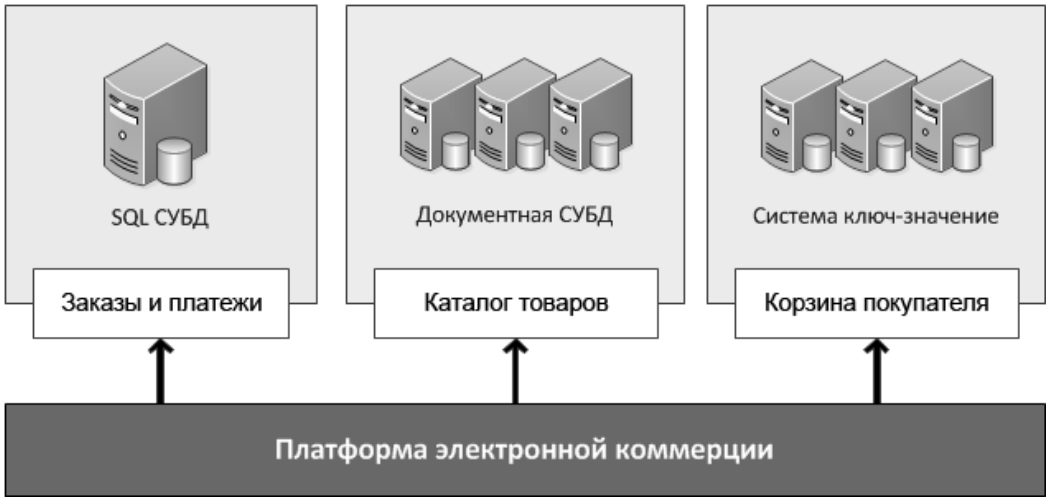

Рис. 2 – Polyglot Persistence

Однако применение такого подхода усложняет приложение, требуя использования различных языков запросов и интерфейсов для доступа к данным. Одним из решений этой проблемы является применение сервис-ориентированной архитектуры (Service-Oriented Architecture, SOA). При этом все детали доступа к данным эффективно скрываются за интерфейсом соответствующего сервиса, позволяя, к тому же, разбить приложение на более или менее независимые части, что упрощает управление разработкой. Этот подход, однако, требует значительных дополнительных усилий и накладных расходов на организацию взаимодействия между сервисами, что делает его применение нецелесообразным для

небольших приложений.

При разработке приложений, использующих системы хранения данных, практически всегда возникает проблема, известная как «impedance mismatch», а именно необходимость каким-либо образом отображать объектную модель приложения на модель данных целевой системы хранения. В случае реляционных СУБД существует множество готовых решений и подходов для осуществления объектно-реляционного отображения (Object-Relational Mapping, ORM). Однако и для NoSQL-решений бывает необходимо осуществлять подобное отображение: например, объектно-документное отображение (Object-Document Mapping, ODM).

Применение готовых решений для объектного отображения, таких как ORM-библиотеки, упрощает и ускоряет разработку приложений, предоставляя высокий уровень абстракции и богатую функциональность. К сожалению, высокий уровень абстракции заметно снижает производительность и делает практически невозможными низкоуровневые оптимизации под конкретную используемую СУБД. По этой причине подобные решения редко применяются в высоконагруженных приложениях, где предпочтительнее использование более оптимизированных под конкретную задачу реализаций (см. [10]).

Слой объектного отображения представляется подходящим для интеграции разнородных систем хранения данных, однако нужно постараться избежать недостатков, присущих традиционным библиотекам объектного отображения. Для этого требуется пересмотр классической архитектуры ORM-библиотек, а именно:

- Использование модульного подхода вместо монолитной системы: слой объектного отображения может быть собран из отдельных «строительных блоков», чтобы лучше отвечать потребностям разработчика. Кроме того, если стандартные реализации компонентов по каким-то причинам не подходят, они должны быть легко заменяемыми.
- Максимально «чистая» объектная модель, свободная от логики взаимодействия с подсистемами хранения. Сущность, в отличие от случая ORM, может содержать не только поля простых типов, но также списки, вложенные объекты и динамические свойства.
- Настраиваемый уровень абстракции: поддержка интерфейсов разных уровней позволяет найти нужный компромисс между уровнем абстракции и производительностью.
- Независимое отображение сущностей и поддержка ассоциаций поверх слоя отображения позволяет иметь целостную объектную модель, несмотря на то, что сущности могут отображаться на разные системы хранения.
- Наличие унифицированного языка запросов с поддержкой базовой функциональности, не привязанного к SQL или другому языку запросов.
- Возможность усилить гарантии целостности данных с помощью валидации на стороне приложения.
- Поддержка кэширования, «ленивой» загрузки, Unit of Work и других успешно себя зарекомендовавших практик (см., например, [11]).

Приведенные принципы были применены при разработке прототипа программного каркаса MapperStack для интеграции разнородных систем хранения данных на уровне объектного отображения в Web-приложениях на PHP. В архитектуре MapperStack можно выделить два основных слоя: слой отображения, состоящий из объектов, реализующих интерфейс Mapper (например, методы getEntityId(), find(), findBy() и т.д.) и верхний слой MapperStack, координирующий работу слоя отображения и предоставляющий возможности интеграции, в том числе поддержку ассоциаций:

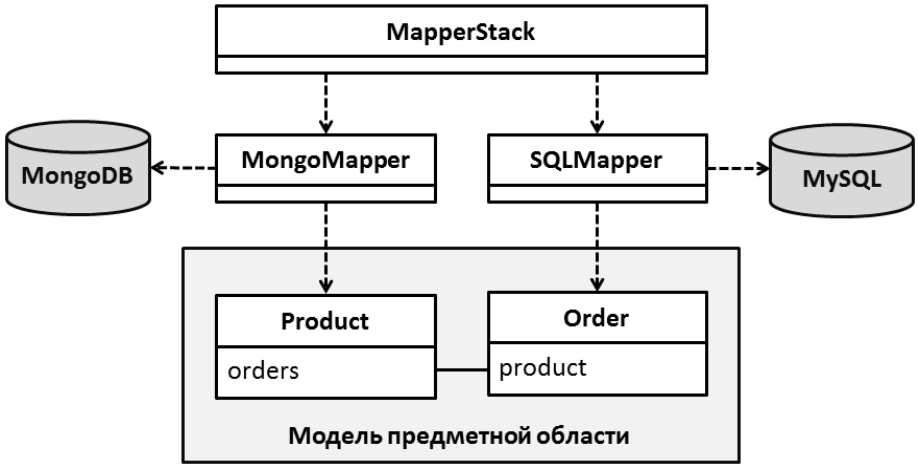

Рис. 3 – Отображение сущностей с помощью MapperStack

Каждый объект в слое отображения ответственен за отображения сущностей определённого класса. Каждая сущность должна иметь уникальный идентификатор (аналог первичного ключа) и может содержать динамические свойства, списки и вложенные объекты. Сущность может содержать метаданные, определяющие правила отображения, например:

```
class BlogPost extends MapperStack\Object
{
  protected $id;
  protected $username;
  protected $text;
  protected $tags;
  protected $comments;

  public static function meta(EntityMetaClass $metaClass)
  {
     $metaClass->db('blog')
               ->collection('posts')
               ->field('id', 'integer', '_id')
               ->field('username', 'string')
               ->field('text', 'string')
               ->field('tags', 'string[]')
               ->object('comments', 'Blog\Comment[]')
               ->id('id');
  }
}
```

В настоящее время реализовано отображение сущностей на MySQL и MongoDB, а также унифицированный язык запросов с трансляцией в SQL и запросы к MongoDB. Поддерживаются операции filter, sort, skip, limit, project и др., условия выборки задаются в виде логического выражения, поддерживаются предикаты (как механизм расширения языка запросов), связываемые и встроенные параметры. Кроме того, если целевая система хранения данных поддерживает документную модель данных, то в запросах можно использовать операции сопоставления элементов коллекции и переход по графу объектов, чтобы добраться до атрибутов вложенных объектов.

Рассмотрим пример запроса в терминах сущностей и результирующие запросы к целевым системам хранения данных:

```
$p = $m->query('Domain\Product')
   ->filter('price >= 1000 && type == "shoes"')
           ->sort('price', 'asc')
   ->limit(3)
   ->getResult();
```

Этот запрос допускает трансляцию как в SQL, так и в язык запросов MongoDB (сопоставление по образцу):

```
SELECT * FROM `products`
  WHERE `price` > 1000
    AND `type` = 'shoes'
  ORDER BY `price`
  LIMIT 3
```

```
db.products.find( {
  $and : [
    { 'price' : { $gte : 1000} },
    { 'type' : 'shoes' }
  ] } ).sort( { 'price' : 1 }
).limit(3);
```

Рассмотрим теперь запрос, содержащий операцию сопоставления элементов массива (в данном случае в массиве comments ищется элемент, для которого выполняется условие в фигурных скобках, кроме того, присутствует предикат регулярного выражения). Такой запрос не может быть выражен средствами SQL, но поддерживается в MongoDB:

```
$posts = $m->query('Blog\BlogPost')
           ->filter('regexp(text, "/programming/i" &&
                comments {username == "ivan"} || rating > 100')
           ->getResult();
```

Результирующий запрос к MongoDB будет выглядеть следующим образом (можно отметить, что запрос в MapperStack является более наглядным и привычным):

```
db.posts.find( { $or : [
             { $and : [
               { 'text' : /programming/i },
               { 'comments' :
                 { $elemMatch: { 'username' : 'ivan' } }
               }
           ] },
           { 'rating' : { $gt : 100 } }
         ] } );
```

Для непосредственной организации отображения в MapperStack реализован ряд компонентов, таких как классы для хранения метаданных, адаптеры для систем кэширования, абстракция отображения типа, механизмы для отслеживания изменений и преобразования данных, Unit of Work (откладывание всех операций модификации данных до вызова метода flush(), синхронизующего состояние с системой хранения и выполняющего операции с учетом ассоциаций [12]) и ряд других компонентов. Каждый компонент может быть легко заменен другой реализацией, кроме того, возможны полностью пользовательские реализации классов отображения. Дальнейшее развитие данного подхода может включать поддержку других типов систем, реализацию оптимистических блокировок, абстракцию MapReduce, поддержку модификации без чтения, агрегатных функций, наследования и т.д.

В заключение рассмотрим некоторые аспекты производительности библиотеки MapperStack на примере модуля объектно-реляционного отображения (SQLMapper). Для этого был разработан тестовый сценарий, включающий выборку сущностей, модификацию полей, удаление сущностей и фиксирование сделанных изменений в базе данных. В качестве эталона приведены характеристики для того же сценария, реализованного с помощью библиотеки Doctrine ORM – мощной и хорошо оптимизированной библиотеки объектно-реляционного отображения для PHP:

Таблица 1. Результаты исполнения тестового сценария

	Время работы (мс)	**Пиковое потребление памяти (Кб)**
MapperStack SQLMapper (первый запуск)	1625	10190
MapperStack SQLMapper (дальнейшие запуски)	595	6940
Doctrine ORM (первый запуск)	1540	9472
Doctrine ORM (дальнейшие запуски)	560	6144

При первом запуске сценария происходит анализ и трансляция запросов с последующим кэшированием, чем и обусловлено несколько большее время работы. Кэш результатов запросов при тестировании не использовался. В плане производительности SQLMapper незначительно уступает специализированным ORM-библиотекам (главным образом, вследствие большей модульности и слабой связанности компонентов программного каркаса). Стоит заметить, что с использованием компонентов MapperStack можно реализовать и более производительное отображение (например, работая напрямую с драйвером СУБД, пропуская этап преобразования типов и т.д.).

Реализация модульного программного каркаса для интеграции различных систем хранения на уровне объектного отображения позволяет быстро разработать прототип приложения, используя стандартную функциональность, а затем при необходимости постепенно заменять их на более эффективные реализации исходя из решаемых задач. Интеграция различных систем хранения открывает новые возможности, позволяя использовать наиболее подходящие решения для работы с данными, сохраняя при этом целостную объектную модель приложения.

Литература

1. D. Merriman, «On Distributed Consistency», 2010. (http://blog.mongodb.org/post/475279604/on-distributed-consistency-part-1)
2. R. Cattell, «Scalable SQL and NoSQL Data Stores», 2011. (http://www.cattell.net/datastores/Datastores.pdf)
3. 451 Research, «NoSQL, NewSQL and Beyond: The drivers and use cases for database alternatives», 2011.
4. С. Д. Кузнецов, А. В. Посконин, «Распределенные горизонтально масштабируемые решения для управления данными», Труды Института системного программирования РАН, т. 24, стр. 327-358, 2013.
5. M. Stonebraker, U. Çetintemel, «"One Size Fits All": An Idea Whose Time Has Come and Gone», ICDE ’05: Proceedings of the 21st International Conference on Data Engineering, Washington, 2005.
6. Wikipedia: «Entity-Attribute-Value model». (http://en.wikipedia.org/wiki/Entity%E2%80%93attribute%E2%80%93value_model)
7. M. Fowler, «Polyglot Persistence», 2011. (http://www.martinfowler.com/bliki/PolyglotPersistence.html)
8. С. Д. Кузнецов, А. В. Посконин, «Возможно ли сотрудничество SQL и NoSQL?», Открытые системы. СУБД, № 9, стр. 38-41, 2013.
9. S. Francia, J. Hileman, «Augmenting RDBMS with MongoDB for eCommerce». (http://www.nosqldatabases.com/main/2011/4/11/augmenting-rdbms-with-mongodb-for-ecommerce.html)
10. А. В. Посконин, «Web-приложения и данные: проблемы абстракции и масштабируемости», Труды Института системного программирования РАН, т. 23, стр. 159-171, 2012.
11. J. Miller, «Design Patterns for Data Persistence», 2009. (http://msdn.microsoft.com/en-us/magazine/dd569757.aspx)
12. M. Fowler, «Patterns of Enterprise Application Architecture», Addison Wesley, 2002.

УДК 004.451.8

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЕ ОПЕРАЦИОННЫЕ СРЕДЫ

Штанюк Антон Александрович, к.т.н., доцент кафедры информационных технологий в предпринимательской деятельности, Нижегородский государственный университет им. Н.И. Лобачевского, Россия, Нижний Новгород, shtan@land.ru

Введение

В прошлой статье [1] автором были затронуты вопросы относительно перспектив развития объектно-ориентированных операционных систем (ОООС). В таких системах сама архитектура предполагает использование объектной парадигмы, а все ресурсы представлены в виде концептуальных объектов. Операционные среды и оболочки являются тесно связанными, но отличными от операционной системы понятиями.

Операционная среда — комплекс программного обеспечения, предоставляющего средства разработки и выполнения прикладных программ.

Этим определением подчёркивается прикладной, пользовательский характер операционной среды, в то время как операционная система выступает лишь одним из её компонентов и концептуально располагается ближе к аппаратному обеспечению ЭВМ. К операционной среде можно добавить и понятие операционной оболочки, которая обеспечивает работу пользовательского интерфейса.

Почему до сих пор не появилось полноценной и массовой объектной операционной системы? Ответить на этот вопрос и легко, и сложно одновременно.

Легко, потому что операционные системы развиваются по законам рынка и конкуренции, они должны удовлетворять запросам пользователей и обладать набором взаимоисключающих показателей: быть быстрыми, но надёжными, стабильными, но переносимыми, компактными, но расширяемыми. Объектная парадигма позволяет создавать хорошие, продуманные архитектуры, но они не обязаны отвечать другим требованиям, особенно в плане скорости работы и компактности.

Сложно ответить потому, что сам характер управления сложными программными системами (а ОС относятся к их числу) предполагает использование объектных компонентов, в то время разработчики настойчиво предлагают архитектуры, далёкие от идеала и с трудом адаптирующиеся к изменчивым требованиям рынка. То есть приходится признавать, что корни проблемы уходят в далёкое прошлое, когда системы создавались без использования объектного подхода, и это наложило отпечаток на дальнейшее развитие. Это относится не только к системам, создававшимся в стенах крупных компаний, но и к тем, что развивались силами сообществ. Чем дальше разработчики уходят от первых шагов по проектированию архитектуры системы, тем меньше у них шансов изменить её в дальнейшем.

Операционная среда позволяет пользователю решать свои задачи с помощью вычислительной машины. Естественно, что при этом задействуются элементы интерфейса, составляющие пользовательское окружение (user environment). Классическое разделение пользовательских интерфейсов на графические (GUI) и командные (CLI) уже не отвечает требованиям времени. В большинстве систем имеются развитые средства поддержки и CLI, и GUI, и пользователю предоставляется право выбора между данными окружениями, а также использования данных средств параллельно. Для пользователя очень важен другой вопрос: а с чем он имеет дело, при использовании команд или графических элементов, что скрывается за «фасадом» окружения? Второй вопрос: насколько расширяемо пользовательское окружение, насколько легко в него добавить новые элементы? И третий вопрос: как соотносится пользовательское окружение и концепции прикладного программирования? И вот здесь нужно признать, что объектный характер операционной среды имеет большое значение и проистекает именно из-за восприятия её пользователем. Попробуем последовательно рассмотреть оба подхода к разработке пользовательского интерфейса, а также подходы к построению системного интерфейса с пользовательскими приложениями.

Командный пользовательский интерфейс

Взаимодействие с пользователем через CLI происходит благодаря командным оболочкам. В Windows это CMD, PowerShell, а в unix-системах – большое семейство shell: bash, csh, zsh.

Наиболее выраженный объектный характер имеет windows-оболочка PowerShell, построенная на классах .NET [2]. При построении файлов сценариев, пользователи и администраторы могут комбинировать классические команды и утилиты с методами классов

.NET. Это особенно ценно, если у пользователей уже накоплен опыт разработки приложений на различных языках программирования, входящих в набор разработчика .NET. Интерфейс с объектами COM, существующими в Windows много лет, также придаёт оболочке объектный характер. Отдельно хочется подчеркнуть возможность встраивания исполняемых компонентов оболочки в другие приложения.

Другие оболочки, зародившиеся много лет назад, не предполагают объектного подхода. Но есть практика использования скриптовых языков (особенно Python или Perl) для решения задач взаимодействия с системой. Python обладает развитыми средствами поддержки ООП и может выступать в качестве языка управления системой и её ресурсами [3]. Многие дистрибутивы Linux распространяются с большим количеством Python-пакетов, позволяющих решать задачи системного администрирования и программирования, а благодаря широкому использованию Python в прикладном программировании, – осуществлять взаимодействие прикладных и системных программ.

Существует несколько альтернативных проектов, мало известных широкой публике, например Pash (как свободная альтернатива PowerShell для многих систем), или Zoid, основанная на Perl. В Сети можно найти ряд материалов, посвящённых объектно-ориентированному программированию в bash.

Графический пользовательский интерфейс

Если перейти от командного интерфейса к графическому, то стоит рассмотреть два наиболее популярных пользовательских окружения unix-систем Gnome и KDE, для которых выпущены фреймворки (GTK+ и QT, соответственно), с помощью которых создаются приложения. Данные окружения сейчас чрезвычайно популярны и используются в большинстве unix-подобных систем. Имеются возможности их использования и в среде Windows, но здесь у них основное назначение состоит в поддержке запускаемых пользовательских приложений, в то время как в unix-системах они ещё и создают полноценную рабочую среду.

Элементы практически всех пользовательских интерфейсов состоят из объектов. Как правило, имеется несколько базовых классов, из которых методом наследования получается разнообразие элементов управления (от диалогов до кнопок и переключателей). Взаимодействие между объектами осуществляется обменом сообщениями, генерированием событий, исключений. При этом наиболее последовательной выступает библиотека QT, а вот в GTK+ практикуется смешение объектного и процедурного стилей.

В мире коммерческих unix-систем имелся положительный опыт разработки объектно-ориентированной операционной оболочки, основанной на библиотеке Motif. Речь идет о первых попытках стандартизации рабочего окружения на многочисленных операционных системах, то есть создание настоящей операционной среды – Common Desktop Environment (CDE) [4]. Эта среда, появившаяся в конце 90-х годов прошлого века, очень активно эксплуатировалась, но была «заморожена» из-за лицензионных ограничений. В настоящее время CDE распространяется с открытой лицензией, и опыт её разработки и эксплуатации мог бы пригодиться современным разработчикам.

Всё чаще для построения программ с GUI используют языки очень высокого уровня и скриптовые (Ruby, Python). Эти языки используют графические библиотеки GTK и QT «напрямую» благодаря специальным «обвязкам», благодаря чему к существующим консольным программам нетрудно разработать полноценный графический интерфейс, не дублируя основное содержание. Эти элементы пользовательского интерфейса также используют в качестве основных элементов объекты.

Интерфейс прикладного программирования

От интерфейса с пользователем обратимся теперь к интерфейсу с приложениями. Большинство операционных систем предоставляют приложениям программный интерфейс (API), который, в основном, представляет собой библиотеки системных функций. Эти

функции сгруппированы по категориям, к ним обычно прилагаются многочисленные структуры данных. Этот интерфейс очень далёк от объектной модели, часто бывает запутанным, нелогичным, избыточным, а также плохо документированным. Разработчики систем всегда оставляют недокументированные средства, благодаря которым могут достраивать систему или управлять её скрытыми настройками.

Своеобразным исключением можно считать безвременно забытую операционную систему BeOS, обладавшую API, реализованным на C++, и предлагавшей концепцию серверов для программных приложений. Идеи, заложенные в BeOS, надолго опередили своё время, и вот теперь их можно встретить в свободной реализации под названием Haiku [5].

Microsoft сделала ряд важных шагов в плане унификации API через прослойку .NET, которая фактически является системой классов и виртуальной машиной, преобразующей вызовы команд CLR в native-инструкции. Благодаря .NET удалось создать немало прикладных программ, имеющих объектный стиль взаимодействия с ресурсами системы. Несомненно, создание .NET было важным маркетинговым и техническим ходом компании, и его развитие было подготовлено предшествующей историей технологии COM.

Выводы

Подводя итоги, мы должны признать, что в операционных средах объектные технологии можно встретить чаще, чем в операционных системах, но и здесь нет общепринятого стандарта на их использование и развитие. Остаётся выразить надежду, что современные активные разработки пользовательских окружений будут учитывать положительный опыт прошлых проектов с использованием ООП, а разработчики системных API будут расширять объектные возможности своих ресурсов.

Литература

1. Штанюк А.А. Перспективы развития объектно-ориентированных операционных систем // Объектные системы – 2013: Материалы VII Международной научно-практической конференции (Ростов-на-Дону, 10-12 мая 2013) – ШИ(Ф) ЮГТУ, 2013, с. 35-39.
2. Коробко И. PowerShell как средство автоматического администрирования. М.: ДМК Пресс, 2014.
3. Гифт Н., Джонс Д. Python в системном администрировании Unix и Linux. М.: Символ-плюс, 2009.
4. CDE 2.1 Data Sheet [Электронный ресурс]. – Режим доступа http://www.opengroup.org/desktop/cde/cde.data.sheet.htm (дата обращения 12.04.2014).
5. Haiku Book [Электронный ресурс]. – Режим доступа https://api.haiku-os.org/ (дата обращения 12.04.2014).

УДК 004.04

ПРЕДМЕТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ СТРУКТУРЫ БАЗЫ ДАННЫХ В ПОНЯТИЯХ МЕТАМОДЕЛИ ОБЪЕКТНОЙ СИСТЕМЫ

Олейник Павел Петрович, к.т.н., Системный архитектор программного обеспечения, ОАО «Астон», Россия, Ростов-на-Дону, xsl@list.ru

Введение

В настоящее время широкую популярность получило предметно-ориентированное проектирование, предполагающее создание модели предметной области, основу которой составляет диаграмма классов, представленная в нотации UML [1-2]. Предполагается, что при реализации информационной системы впоследствии применяется объектно-ориентированный язык программирования, а для хранения информации в долговременной памяти – используется СУБД. При проектировании структуры БД можно выделить три основных этапа: концептуальное, логическое и физическое проектирование [3]. Последний

этап напрямую зависит от типа выбранной СУБД, поэтому при реализации приложения в инструментальной среде разработки, охватывающий полный жизненный цикл, возникают сложности при построении физической модели. В статье рассматривается только статическая структура, представляющая, по сути, диаграмму классов.

1. Структура метамодели объектной системы

Процесс проектирования рассматривается на основе метамодели, первоначально опубликованной в работе [4]. Текущая версия метамодели объектной системы представлена на рисунке 1 и имеет следующие ключевые отличия от версии, описанной в [4]:

1. Добавлены классы, представляющие атрибуты различных типов данных такие как геометрические объекты (точки, линии, замкнутые фигуры), географические координаты и атрибуты, позволяющие представлять экземпляры классов.
2. Добавлены вычисляемые классы в систему, экземпляры которых не сохраняются в базе данных и используемые для реализации алгоритмов и бизнес-процессов прикладной предметной области.
3. Спроектирована и реализована иерархия классов, представляющая значения по умолчанию для атрибутов классов. При этом имеется возможность наследования значений и переопределения значений в производных классах (полиморфизм).
4. Добавлены различные опции (в виде множества) для классов, позволяющие управлять поведением экземпляров. Например, имеется возможность указать, что класс реализует шаблон проектирования Singleton и тем самым позволяет запретить возможность создания более одного экземпляра и возможность удаления.

2. Основные этапы проектирования структуры базы данных

Перейдем к процессу предметно-ориентированного проектирования с позиций разработки структуры БД. Выделяют три основных этапа проектирования: концептуальное, логическое и физическое [3]. Концептуальное проектирование базы данных позволяет описать высокоуровневую модель и начинается с создания концептуальной модели данных предприятия, полностью независимой от любых деталей реализации. К последним относятся выбранный тип СУБД, состав программ приложения, используемый язык программирования, конкретная аппаратная платформа, вопросы производительности и любые другие физические особенности реализации. Для целей данной статьи рассмотрим простую предметную область, концептуальная и логические модели которой представлены на рисунке 2.

Следующим этапом является логическое проектирование БД, под которым понимается конструирование информационной модели предприятия на основе существующих конкретных моделей данных, но без учёта специфики используемой СУБД и прочих условий реализации [3].

В общем случае логическое проектирование базы данных заключается в преобразовании концептуальной модели данных в логическую модель данных предприятия с учетом выбранного типа СУБД (в нашем случае, в понятия объектной СУБД). Логическая модель данных является источником информации для этапа физического проектирования. Она предоставляет разработчику физической модели данных средства проведения всестороннего анализа различных аспектов работы с данными, что имеет важное значение для выбора эффективного проектного решения.

При разработке логической структуры (модели) БД используется объектно-ориентированная модель данных (ООМД), поэтому необходимо удалить элементы, присутствующие на концептуальной модели, которые не поддерживаются в ООМД и добавить те элементы, которыми отличается ООМД. В нашем случае необходимо выполнить:

1. Обобщение/специализацию.
2. Преобразовать связи с атрибутами в отдельные классы.

3. Преобразование сложных связей в отдельные классы.
4. Проектирование правил поведения.

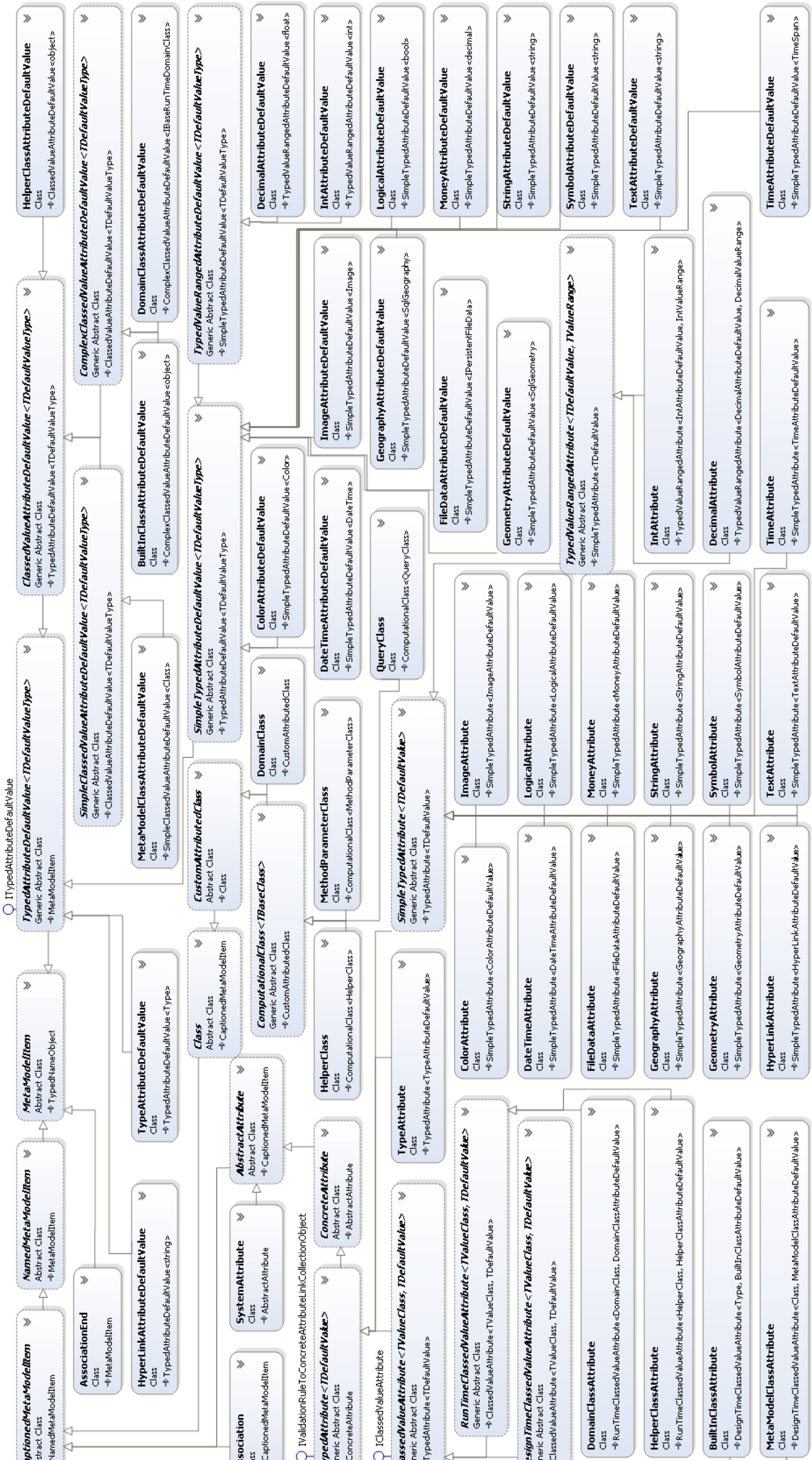

Рис. 1 – Метамодель объектной системы

Рассмотрим каждый пункт более подробно. Некоторые объекты могут иметь подобные, но не идентичные атрибуты и методы. Если степень такого подобия высока, то имеет смысл совместно использовать эти свойства (атрибуты и методы). Наследование позволяет определять один класс на основе общего класса-предка. Такие менее общие классы называются подклассами (производными, унаследованными классами), а более общие — суперклассами (родительскими классами). Процесс образования суперкласса называется обобщением (генерализация), а процесс образования подкласса — специализацией. По умолчанию подкласс наследует все свойства суперкласса и в дополнение к ним может определить свои собственные уникальные. В подклассе могут быть переопределены унаследованные свойства. Все экземпляры подкласса являются также экземплярами суперкласса и, согласно принципу подстановки, для любого метода и конструкции вместо экземпляра суперкласса всегда можно использовать экземпляр его подкласса. Отметим, что расширенная модель «сущность-связь» поддерживает механизм уточнения/обобщения (предполагающий использование дискриминанта), но данная конструкция более ограничена по сравнению с концепцией наследования, применяемой в объектно-ориентированном программировании.

Проанализировав рис. 2, можно утверждать, что два выделенных класса (Client и Product) полностью идентичны, так как характеризуются одинаковыми атрибутами (в данном случае названием). Поэтому имеет смысл выделить общий базовый класс (выполнить процедуру обобщения), содержащий название (например, класс NamedObject), а классы Client и Product сделать производными от NamedObject.

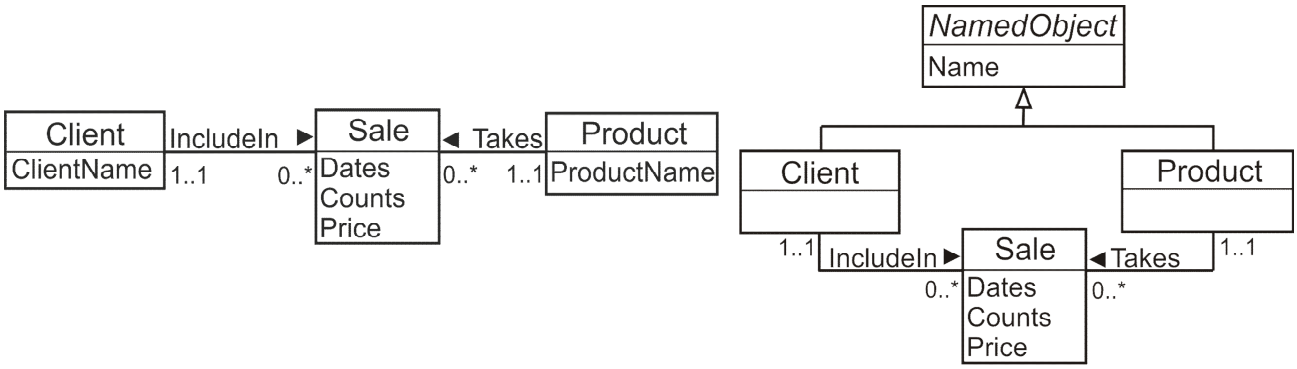

Рис. 2 – Концептуальная (слева) и логическая (справа) модели

В сложных связях участвуют три и более сущностей. Такие связи не поддерживаются непосредственно и требуют введения промежуточного класса, который соединяется с остальными бинарными связями (как правило, с типом «один ко многим»). Из концептуальной модели видно, что в предметной области отсутствуют сложные связи, поэтому описанные преобразования выполнять не нужно. Результат описанных действий представлен на рис. 2.

Последним этапом является физическое проектирование базы данных, под которым понимается описание конкретной реализации базы данных, размещаемой во внешней памяти. Физический проект описывает базовые отношения, определяет организацию файлов и состав индексов, применяемых для обеспечения эффективного доступа к данным, а также регламентирует все соответствующие ограничения целостности и меры защиты. Физическое проектирование базы данных предусматривает принятие разработчиком окончательного решения о способах реализации создаваемой базы. Поэтому физическое проектирование обязательно производится с учетом всех особенностей целевой СУБД.

3. Физическая модель структуры БД в понятиях метамодели объектной системы

В нашем случае используется собственная среда разработки, поэтому физическая модель представляется в понятиях экземпляров классов метамодели, изображенной на рис. 1.

Т.е. необходимо построить диаграмму объектов языка UML [2]. Один из возможных вариантов представлен на рис. 3.

Рассмотрим представленный рисунок более подробно. Из рисунка видно, что представлена диаграмма объектов языка UML [2], содержащая некоторую дополнительную информацию, необходимую для понимания. Для представления классов предметной области используются объекты типа DomainClass. Для организации ассоциаций используются объекты трех классов. Объекты типа Association позволяют описать ассоциации между классами. В настоящий момент используются только бинарные ассоциации (сложные ассоциации могут быть представлены в виде набора бинарных ассоциаций) поэтому необходим механизм описания обоих краев ассоциации. Для этого используются экземпляры классов AssociationEnd. Для каждого края ассоциации генерируется атрибут типа DomainClassAttribute.

В зависимости от типа (домена) атрибута класса используется соответствующий экземпляр класса метамодели объектной системы. Так, для атрибута Dates, позволяющего сохранять дату и время продажи, использован экземпляр класса DateTimeAttribute. Из рисунка видно, что подобным образом может быть представлена модель любой предметной области.

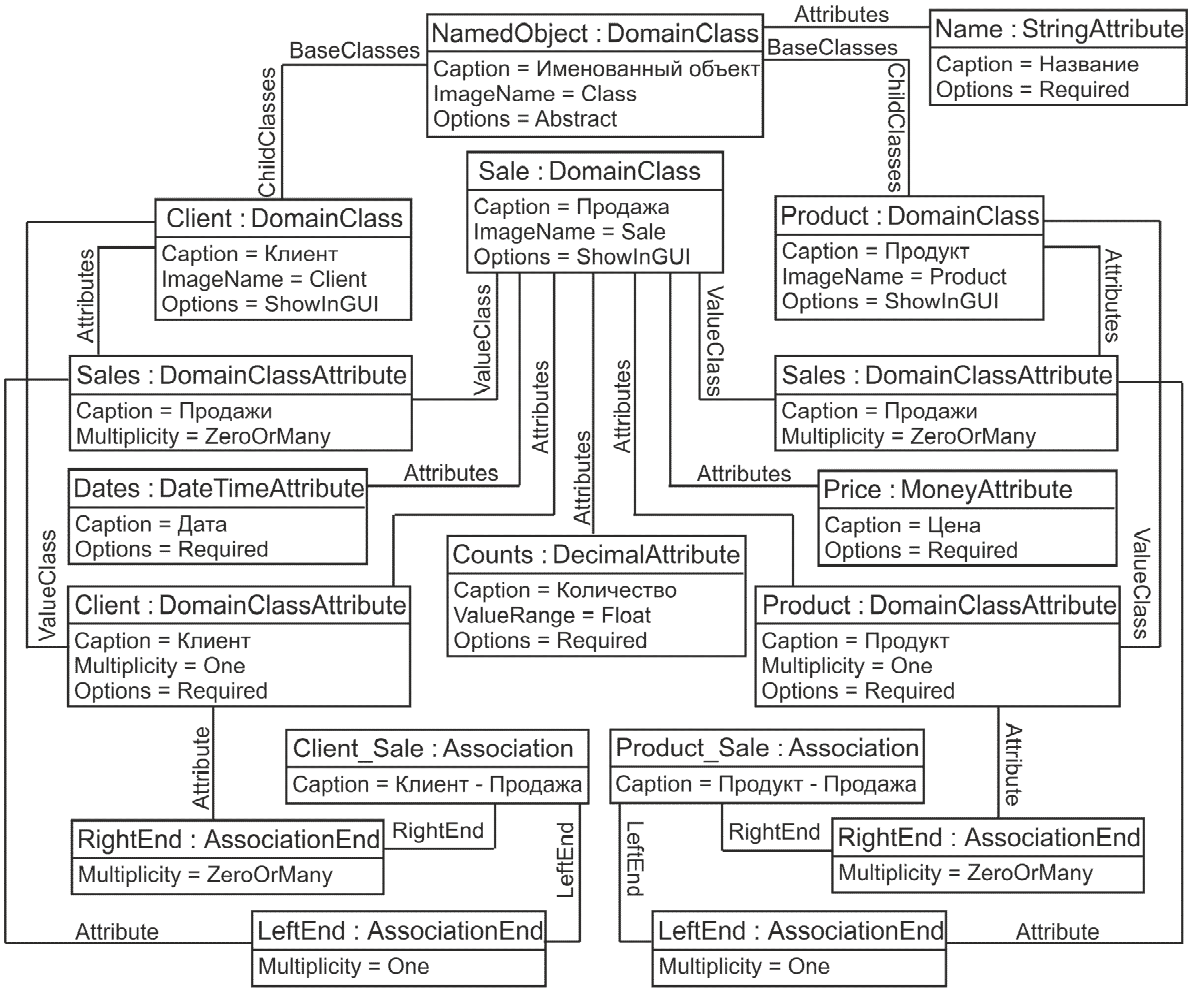

Рис. 3 – Физическая модель структуры БД в понятиях метамодели объектной системы

Выводы

В данной статье подробно рассмотрен процесс предметно-ориентированного проектирования структуры БД тестовой предметной области. Предполагается, что физическая модель структуры базы данных изображается в виде диаграммы объектов языка UML. Классы отображаемых экземпляров представляются классами метамодели объектной

системы. В будущем предполагается реализовать механизм автоматического построения физической модели по имеющимся экземплярам классов. Кроме того, необходимы механизмы представления поведения системы, т.е. ее динамической составляющей.

Литература

1. Эрик Эванс. Предметно-ориентированное проектирование (DDD). Структуризация сложных программных систем : Пер. с англ. – М.: Издательский дом "Вильямс", 2010. – 448 с. : ил. – Парал. тит. англ.
2. Новиков Ф.А., Иванов Д.Ю. Моделирование на UML. Теория, практика, видеокурс. – СПб.: Профессиональная литература, Наука и Техника, 2010. – 640 с.: ил. + цв. Вклейки (+ 2 DVD).
3. Коннолли, Томас, Бегг, Каролин. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание. : Пер. с англ. – М.: Издательский дом "Вильямс", 2003. – 1440 с. : ил. – Парал. тит. англ.
4. Олейник П.П. Иерархия классов метамодели объектной системы // Объектные системы – 2012: материалы VI Международной научно-практической конференции (Ростов-на-Дону, 10-12 мая 2012 г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ ЮРГТУ (НПИ), 2012. – С. 37-40

УДК 681.3

ВИЗУАЛИЗАЦИЯ ОНТОЛОГИЧЕСКИХ МОДЕЛЕЙ И СТАТИЧЕСКИХ ДАННЫХ

Мухутдинов Руслан Маисович, студент, ГОУ ВПО «Уральский федеральный университет имени первого Президента России Б.Н.Ельцина», Нижнетагильский технологический институт (фил.), Факультет экономики и менеджмента, кафедра информационных технологий, Россия. Нижний Тагил, spider9323@gmail.com

Шишкина Вероника Валерьевна, студент, ГОУ ВПО «Уральский федеральный университет имени первого Президента России Б.Н.Ельцина», Нижнетагильский технологический институт (фил.), Факультет экономики и менеджмента, кафедра информационных технологий, Россия. Нижний Тагил

Грегер Сергей Эдуардович, доцент, ГОУ ВПО «Уральский федеральный университет имени первого Президента России Б.Н.Ельцина», Нижнетагильский технологический институт (фил.), Факультет экономики и менеджмента, кафедра информационных технологий, Россия. Нижний Тагил, segreger@gmail.com

Предоставление провайдерами возможностей беспроводного доступа к сети Интернет, расширение каналов связи играет важную роль в развитии современных информационных технологий. Потребители любого вида информационных услуг хотят получать доступ к необходимым сервисам в любое время и в любом месте. В связи с этим, важным этапом в развитии информационных технологий любой сложности и уровня (будь то база данных клиентов банка или графический редактор для художника) является их переход на веб-интерфейс, развитие распределённых систем, облачных инфраструктур.

С быстрым ростом численности населения, с вовлечением новых поколений в различные сферы деятельности, требующие постоянно растущего уровня знаний, умений и навыков, появились онтологические модели. Теперь информация и знания стали структурированы и описаны таким образом, чтобы получатель (клиент) был способен понять и текст, и контекст сообщения.

Автор рассмотрел проблему визуализации онтологических моделей и статистических данных посредством современных веб-технологий. Накопление и анализ данных, представление наглядных графиков — достаточно распространённая задача. Составление прогноза погоды, отображение статистики посещаемости веб-ресурсов, анализ данных любого технического производства (например, на предмет установления частоты появления бракованных изделий) — примеры областей и задач для которых необходимо отображать наглядные графики.

Наличие программного продукта, который предоставляет клиентам возможность анализировать статические данные и онтологические модели, строить необходимые графики, решает ряд важных задач.

Во-первых, визуализация накопленных знаний и данных. Во-вторых, их анализ. В-третьих, получение доступа к функционалу данного продукта в любое время и в любом месте. В-четвёртых, нагрузка по обработке данных распределяется между персональным компьютером пользователя и сервером, за счёт этого снижается нагрузка на машину клиента. В-пятых, пользователь может использовать функционал данного продукта не только на персональном компьютере, но и на любом другом устройстве, которое имеет доступ к сети интернет. В-шестых, вся ответственность по обеспечению качества услуг, сокрытию информации от третьих лиц, защите данных от вредоносных программ и шпионов ложится на поставщика услуги.

Однако существующие на данный момент программные продукты решают вопрос визуализации данных в виде графиков применительно к конкретной области. К примеру, «Google Analytics» [1] или «Яндекс метрика» [2] предназначены для отображения информации о посещаемости веб-сайтов. Область применения накладывает свои ограничения, среда становится узконаправленной. А существующий на данный момент программный продукт «OntoEditor» [3], который предоставляет возможности создания онтологических моделей, хотелось бы дополнить визуализацией этих моделей.

Комплексное решение на базе веб-технологий, которое могло бы работать с любыми входными данными, записанными в разных форматах, и предоставляющее функционал по обработке статических данных и накопленных знаний, стало бы достойной альтернативой программам, работающим на стационарных машинах.

Научно-исследовательская работа «Визуализация онтологических моделей и статических данных» посвящена созданию и внедрению такого решения. Целью работы стала разработка программного продукта для обработки статистической информации и накопленных знаний, предоставления наглядных графиков для визуализации данных и онтологических моделей.

Помимо оговоренных выше задач, разрабатываемый программный продукт должен обладать следующими техническими возможностями:

1. поддержка форматов, в которых могут быть поданы данные (ontology, xml, json, csv и др.);
2. поддержка построения сложных графиков (дерево, графы, дерево отображений или treemap);
3. поддержка настройки графиков;
4. дружественный интерфейс.

Поскольку в качестве программ для отображения графиков выступают веб-браузеры, было необходимо определить возможные подходы к вставке графических данных в html-страницу (веб-страницу). Здесь можно использовать либо вставку в страницу сторонних плагинов, либо использовать Html5 и JavaScript. Было решено отказаться от сторонних плагинов в пользу использования языка JavaScript и библиотеки JIT [4]. В качестве среды для организации серверной части разрабатываемого решения был выбран framework Plone [5] и язык Python, которые также распространяются свободно.

На основе поставленных задач и выбранной среды разработки можно смоделировать систему классов посредством языка UML [6]. На рис.1 представлена диаграмма классов.

Так как для отображения данных была выбрана библиотека JIT, то данная библиотека накладывает свою специфику на формат входных данных для построения графиков. Это json-формат, у которого есть обязательные свойства: id – номер узла, name – имя узла, children – дочерний элемент узла. Таким образом, все входные данные необходимо преобразовывать в json-формат с особой структурой. Данным преобразованием и построением занимаются следующие четыре класса: toJSON, Collapse, BuildObject, Paint.

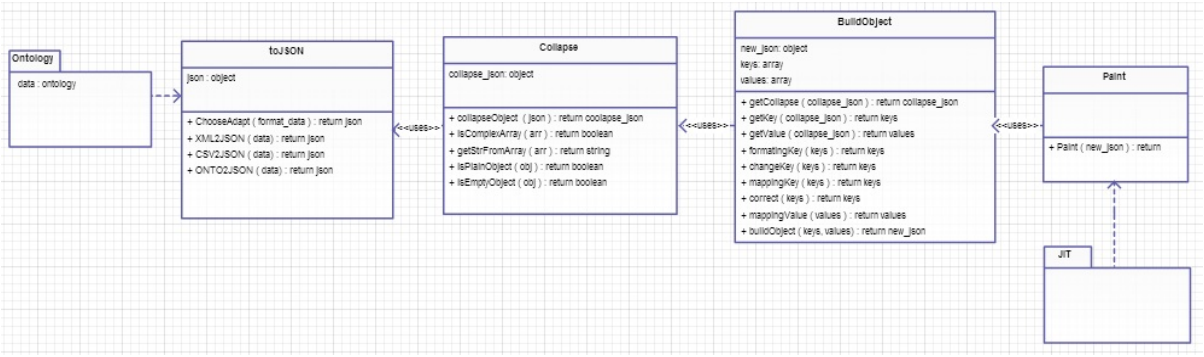

Рис. 1 – Диаграмма классов

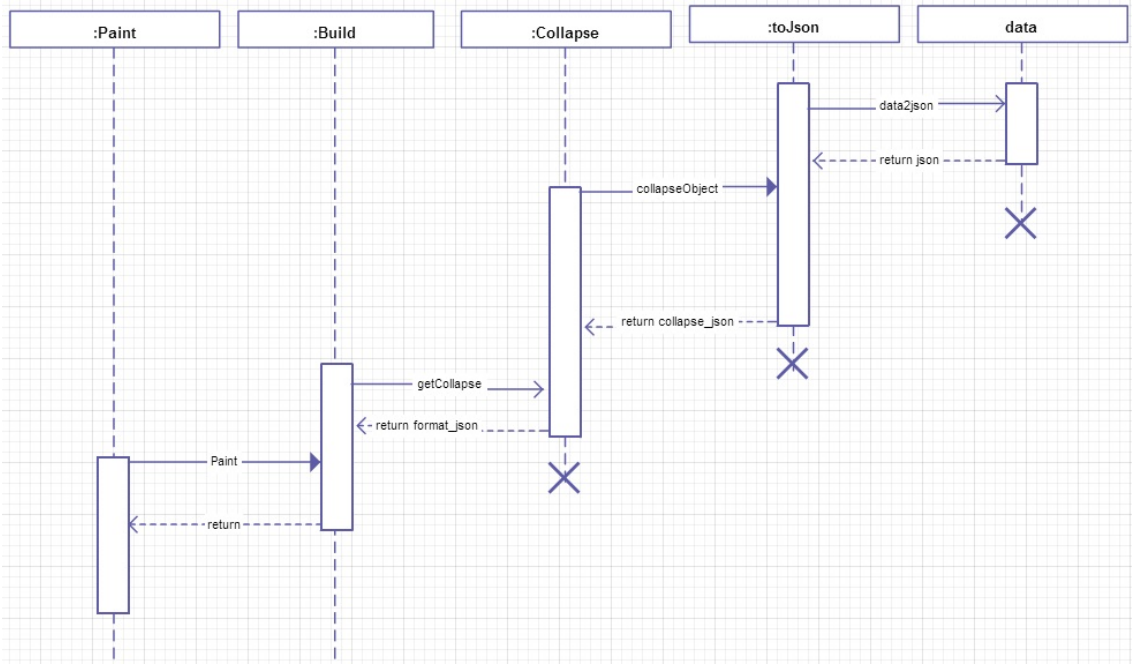

Рис. 2 – Диаграмма последовательности

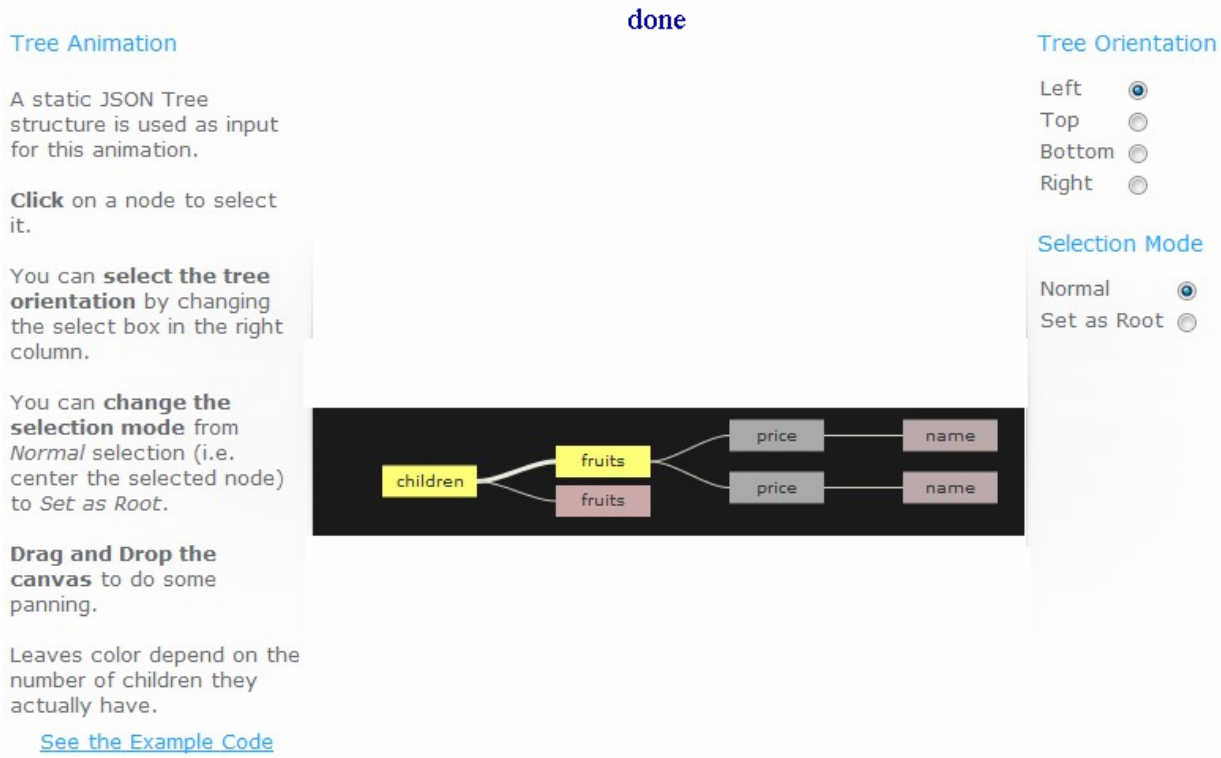

Рис. 3 – Пример визуализации онтологической модели

Функционал класса toJSON заключается в том, чтобы входные данные конвертировать в данные формата json и затем хранить их. Классы Collapse и BuildObject преобразуют данные, хранящиеся в классе toJSON, в данные формата json, учитывая особенности библиотеки JIT, и хранят эти данные. Класс Paint на основе преобразованных данных и средств библиотеки JIT строит графики. То, как преобразуются входные данные во времени, можно увидеть на диаграмме последовательности, рис.2.

Были изучены возможности библиотеки JIT и реализован функционал по преобразованию данных: конвертация входных данных в json-формат, преобразование данных в json-формат с особой структурой данных. Был реализован функционал рисования графиков.

Пример вывода графика представлен на рис. 3.

Дальнейшие направления исследований включают в себя:

- Добавление новых форматов входных данных.
- Размещение в сети Интернет, тестирование программного продукта.
- Добавление визуализации онтологических моделей в продукт «OntoEditor».

Литература

1. Официальный сайт Google Analytics. – [электронный ресурс]. – Режим доступа: http://www.google.com/analytics/
2. Официальный сайт Яндекс-метрика. – [электронный ресурс]. – Режим доступа: https://metrika.yandex.ru/
3. Официальный сайт построения онтологических моделей. – [электронный ресурс]. – Режим доступа: http://www.ontoprojects.ru/
4. Официальный сайт библиотеки JIT. – [электронный ресурс]. – Режим доступа: http://philogb.github.io/jit/
5. Официальный сайт Plone. – [электронный ресурс]. – Режим доступа: http://www.prone.org/
6. Официальный сайт UML. – [электронный ресурс]. – Режим доступа: http://www.uml.org/
7. Jacob Gube. 20 Frash JavaScript Data Visualization Libraries. – [электронный ресурс]. – Режим доступа: http://sixrevisions.com/javascript/20-fresh-javascript-data-visualization-libraries/
8. Steve Jobs. Thoughts on Flash. – [электронный ресурс]. – Режим доступа: http://www.apple.com/hotnews/thoughts-on-flash/
9. Грегер С.Э. Редактор метамодели онтологической системы // Объектные системы – 2012: материалы VI Международной научно- практической конференции (Ростов-на-Дону, 10-12 мая 2012 г.) / Под общ. ред. П.П. Олейника. — Ростов-на-Дону: ШИ ЮРГТУ (НПИ), 2012. — с. 88-92.

УДК 004. 891

РАЗРАБОТКА МОДУЛЯ УПРАВЛЕНИЯ УРОВНЕМ ОСВЕЩЁННОСТИ ЛОКАЛЬНЫХ ЗОН МНОГОФУНКЦИОНАЛЬНОГО ПОМЕЩЕНИЯ НА ОСНОВЕ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПОДХОДА

Сергеев Евгений Владимирович, студент, Комсомольский-на-Амуре государственный технический университет, Россия, Комсомольск-на-Амуре, eugenejay@mail.ru
Горькавый Михаил Александрович, канд. техн, наук, доцент, Комсомольский-на-Амуре государственный технический университет, Россия, Комсомольск-на-Амуре, idpo@knastu.ru

Введение

На сегодняшний день энергосбережение является одной из самых актуальных проблем энергетики. Многие учёные занимаются вопросом о разработке новых источников энергии и более эффективных способов её использования. В настоящее время правительство очень заинтересованно в данной проблеме, созданы органы управления в этой отрасли и нормативные документы. Одним из самых распространённых способов экономии

электроэнергии является оптимизация потребления электроэнергии на освещение. Существует множество рекомендаций и мероприятий для решения этой проблемы. Таких, как максимальное использование дневного света, оптимальное размещение световых источников, замена ламп накаливания на люминесцентные и др. Одним из актуальных решений этой проблемы является установка интеллектуальных систем управления освещением.

В ФГБОУВПО «КнАГТУ» в ЛПИП ведутся разработки интеллектуальной системы управления, основанной на оценке и сравнении сигналов естественного и искусственного освещения.

Основные задачи:

- Создание модели помещения.
- Расчёт КЕО, %.
- Расчёт искусственного освещения.
- Классификация локальных зон помещения.
- Создание Интеллектуального блока управления.
- Проверка и отладка системы.

Создание модели помещения

Модель необходима для тестирования интеллектуальной системы энергопотребления.

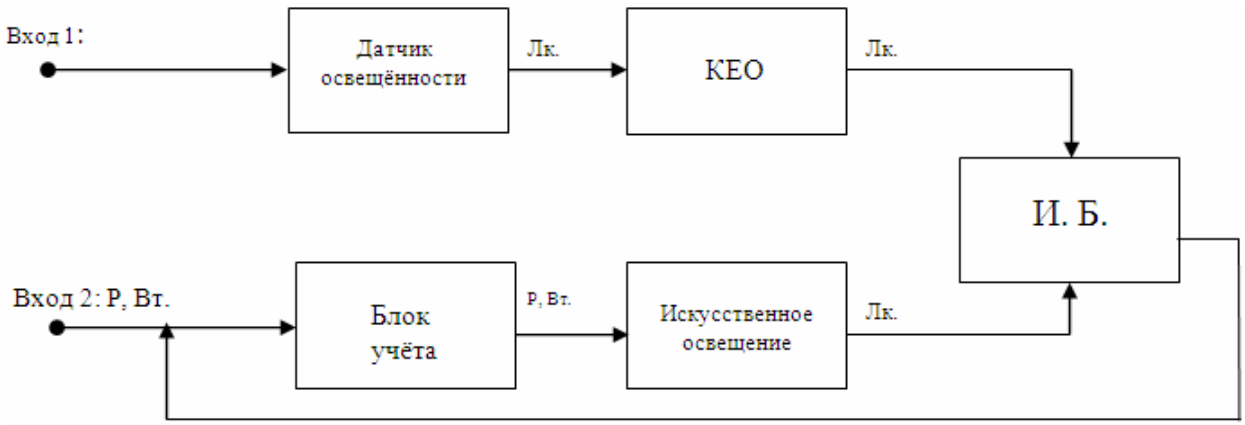

Рис. 1 – Упрощённая схема модели интеллектуальной системы управления

На представленном рисунке представлены следующие блоки с установленными взаимосвязями:

- блок «датчик освещённости» – это набор измерительных устройств для сбора данных об освещённости на улице;
- блок КЕО – предназначен для расчёта коэффициентов естественного освещения в локальных зонах внутри помещения;
- блок учёта – выполняет функцию счётчика;
- блок «искусственное освещение» – предназначен для расчёта коэффициентов искусственного освещения в локальных зонах внутри помещения;
- И. Б. – это интеллектуальный блок, предназначенный для повышения эффективности энергопотребления. Подробнее об интеллектуальном блоке в работе [2].

На вход 1 системы приходит сигнал с датчика, измеряющий освещённость на улице, и входит в Блок с рассчитанным коэффициентом естественной освещённости. На выходе из Блока КЕО мы получаем освещённость в каждой из локальных зон. На вход 2 поступает мощность и проходя через блок учёта мощности попадает в блок с рассчитанными коэффициентами искусственного освещения, на выходе получаем освещённость в локальной зоне, создаваемую искусственным освещением. После чего сигналы попадают в

интеллектуальный блок, где они сравниваются и производится необходимое регулирование. Для качественного регулирования сигналы сравниваются в локальных зонах.

Определение локальной зоны на основе объектно-ориентированного подхода

Работа технических устройств будет основываться на интеллектуальном алгоритме [2]. Каждая из локальных зон наследует следующие атрибуты субъекта управления ЭВМ с установленным ПО: такие как фунциональные возможности, класс точности и функциональное назначение. Кроме того, субъект управления использует данные о локальных зонах: такие как расположение локальной зоны от окна, данные о расположении искуственного освещения над локальной зоной и информацию о виде работ.

Для создания диаграммы классов применялась общепринятая система обозначений [1].

На данной диаграмме изображены классы локальных зон. Наличие естественного освещения – определяет данные о расположении локальной зоны от светового проёма, т.е. расположена локальная зона близко либо далеко от окна.

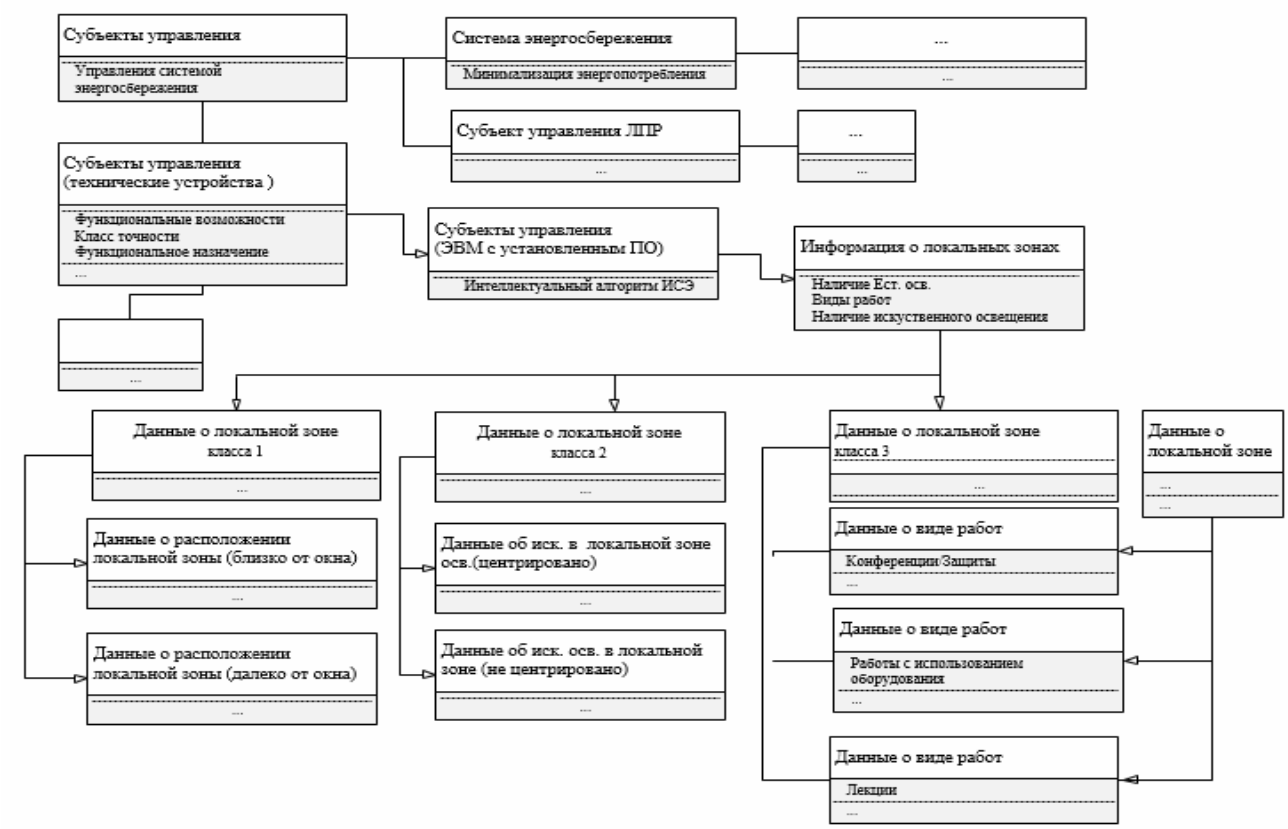

Рис. 2 – Диаграмма классов

Наличие искусственного освещения – определяет наличие искусственного освещения над локальной зоной. Центрированное искусственное освещение – это когда источник света будет находиться непосредственно над локальной зоной. Такое расположение источника света будет способствовать качественному регулированию.

Вид работ – определяет, какая работа будет осуществляться в многофункциональном помещении. От этого класса будет зависеть выбор разряда зрительной работы и определение количества локальных зон.

Вид работ – будет выбираться в зависимости от рода предприятия или организации и его специфики.

Диаграмма классов будет использоваться в интеллектуальном блоке. Она будет определять локальные зоны.

Заключение

Разработанная диаграмма классов для определения локальных зон помещения позволяет точно определить количество локальных зон, необходимых в регулировании освещения. Она позволяет полноценно оценить факторы, влияющие на работу системы. По данной диаграмме можно определить локальные зоны в любом помещении, так как классы, выбранные в качестве характеристик помещений, являются унифицированными.

Литература

1. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ / Г. Буч; пер. с англ. И. Романовского; под ред. Ф. Андреева – М.: Невский Диалект, 2000. – 359 с.
2. Болдырев, В.В. Концепция интеллектуального алгоритма автоматизированной системы энергопотребления/ В.В. Болдырев, М.А. Горькавый // Технические и математические науки: актуальные проблемы и перспективы развития: материалы междунар. науч.-практ. конф., Саратов, 14.11.2013г., С. 19 – 24.

УДК 004.43:378.09

СЕРИАЛИЗАЦИЯ И ДЕСЕРЕАЛИЗАЦИЯ ОБЪЕКТОВ PYTHON В YAML – ФОРМАТ

Мясникова Нелли Александровна, доцент, Южно-Российский государственный политехнический университет (НПИ), Россия, Новочеркасск, mnela@list.ru

Шепилов Владислав Александрович, студент, Южно-Российский государственный политехнический университет (НПИ), Россия, Новочеркасск, ship3000@mail.ru

Компания «Delmar» торгует ювелирными изделиями и сотрудничает с таким компаниями, как Sterling, Walmart, Commercehub, Ebay, Zulily, и многими другими. В своей деятельности для внутренних бизнес-процессов эта компания использует CRM-систему на базе OpenERP 6.1. Главной задачей для обеспечения работоспособности системы является унификация входных данных от покупателей. Система OpenERP написана на языке программирования Python 2.7, дополнительные модули, специфичные для компании «Delmar», также написаны на языке программирования Python. Поэтому в статье унификация входных/выходных данных и примеры будут рассмотрены для этого языка.

Упрощенная схема бизнес-процесса покупки показана на рисунке 1.

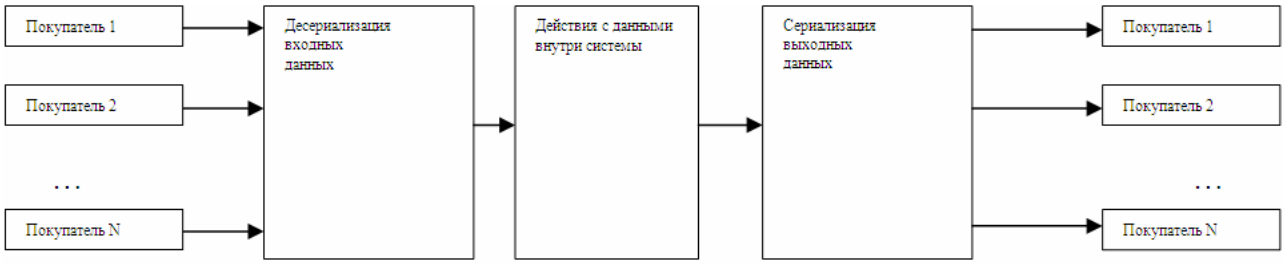

Рис. 1 – Упрощенная схема бизнес-процесса покупки

При разработке программного обеспечения часто возникает необходимость передачи данных между разными программными продуктами/частями программного продукта/сброса данных в базу данных. Сериализация – процесс перевода какой-либо внутренней (для языка программирования) структуры данных во внешнее представление в каком-либо из общеиспользуемых форматов обмена данными. Обратной к операции сериализации является операция десериализации (структуризации) – восстановление начального состояния структуры данных из внешнего представления[4].

За время эволюции компьютерной техники и программного обеспечения появилось множество человекочитаемых форматов данных и языков разметки. Наиболее известные: XML, YAML, JSON. Начиная с исследования форматов, следует учитывать, что

унифицированное представление входных данных должно без проблем проверяться человеком на правильность. Рассмотрим более подробно каждый из форматов.

```
Пример конечного объекта, используемого в последующем в объекте внутри
системы:
order = {
    "order_id": "PO453234",
    "ship": {
        "address": "11345 Colmar Street",
        "city": "Dallas",
        "country": "USA",
        "name": "Camela Yong",
        "phone": 00000000000,
        "state": "NY",
        "zip": 342, },
    "lines": [
        {
            "carrier": "UPS",
            "sku": "BRK3144",
            'qty': 1,
            "cost": 35.0,
        }, ], }
```

XML

XML (англ. eXtensible Markup Language – расширяемый язык разметки) – рекомендованный Консорциумом Всемирной паутины (W3C) язык разметки. Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержимому). XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком, с подчёркиванием нацеленности на использование в Интернете [5].

```
Пример объекта order в XML
<?xml version="1.0"?>
<!DOCTYPE PyObject SYSTEM "PyObjects.dtd">
<PyObject family="obj" type="builtin_wrapper"  class="_EmptyClass">
<attr name="__toplevel__" type="dict" id="15692080" >
  <entry>
    <key type="string" value="order_id" />
    <val type="string" value="PO453234" />
  </entry>
  <entry>
    <key type="string" value="ship" />
    <val type="dict" id="15711664" >
      <entry>
        <key type="string" value="city" />
        <val type="string" value="Dallas" />
      </entry>
      <entry>
        <key type="string" value="name" />
        <val type="string" value="Camela Yong" />
      </entry>
      <entry>
        <key type="string" value="zip" />
        <val type="numeric" value="342" />
      </entry>
      <entry>
        <key type="string" value="country" />
        <val type="string" value="USA" />
      </entry>
      <entry>
        <key type="string" value="phone" />
```

```
          <val type="numeric" value="0" />
        </entry>
        <entry>
          <key type="string" value="state" />
          <val type="string" value="NY" />
        </entry>
        <entry>
          <key type="string" value="address" />
          <val type="string" value="11345 Colmar Street" />
        </entry>
      </val>
    </entry>
    <entry>
      <key type="string" value="lines" />
      <val type="list" id="140426501577760" >
        <item type="dict" id="15766800" >
          <entry>
            <key type="string" value="sku" />
            <val type="string" value="BRK3144" />
          </entry>
          <entry>
            <key type="string" value="carrier" />
            <val type="string" value="UPS" />
          </entry>
          <entry>
            <key type="string" value="cost" />
            <val type="numeric" value="35." />
          </entry>
          <entry>
            <key type="string" value="qty" />
            <val type="numeric" value="1" />
          </entry>
        </item>
      </val>
    </entry>
  </attr>
  </PyObject>
```

Плюсы использования XML:

- практически повсеместная поддержка;
- поддерживается на низком аппаратном, микропрограммном и программном уровнях в современных аппаратных решениях.

Минусы использования XML:

- плохая человекочитаемость;
- большой размер.

Конкретно в языке Python присутствует несколько библиотек для работы с XML, и сериализация/десереализация объекта представляется не совсем тривиальной задачей или связана с использованием сторонних библиотек.

JSON

JSON (англ. JavaScript Object Notation) – текстовый формат обмена данными, основанный на JavaScript и обычно используемый именно с этим языком.

```
Пример объекта order в JSON
{"order_id": "PO453234", "ship": {"city": "Dallas", "name": "Camela Yong",
"zip": 342, "country": "USA", "phone": 0, "state": "NY", "address": "11345
Colmar Street"}, "lines": [{"sku": "BRK3144", "carrier": "UPS", "cost":
35.0, "qty": 1}]}
```

Основная проблема использования JSON внутри языка Python сопряжена с тем, что JSON поддерживает только базовые типы str, unicode, int, long, float, bool, None. Поэтому при

сериализации объекта в JSON не базовые типы будут вызывать исключение TypeError, а при десериализации не смогут быть однозначно интерпретированы.

Плюсы:

- распространён;
- поддержка в любом браузере.

Минусы:

- поддержка только базовых типов Python.

YAML

YAML – человекочитаемый формат сериализации данных, концептуально близкий к языкам разметки, но ориентированный на удобство ввода-вывода типичных структур данных многих языков программирования. Изначально YAML являлось акроним от словосочетания Yet Another Markup Language(Ещё один язык разметки), но впоследствии YAML преобразовался в человекочитаемый формат данных и поэтому акроним теперь означает YAML Ain't Markup Language(YAML – не язык разметки). Ко всему прочему, начиная с версии 2.1, JSON является валидным объектом для сериализации/десериализации через YAML-библиотеку[3].

Налицо видны очевидные плюсы этого формата данных, можно добавить только то, что из всех форматов данных YAML является наиболее человекочитаемым. Напишем класс для более простой работы с YAML в Python.

```
Содержание файла yaml_class.py:
from yaml import dump as dump_data
from yaml import load as load_data
from os import access, F_OK, W_OK, R_OK
import logging

_logger = logging

class _YAML():
    def __init__(self, work_with_file=None):
        self.filename = work_with_file
    def deserialize(self, _data=None):
        result = None
        if(self.filename is not None):
            if not access(self.filename, F_OK):
                _logger.error("no find file: %s!" % self.filename)
                result = None
            else:
                if access(self.filename, R_OK):
                    with open(self.filename, "r") as config:
                        result = load_data(config.read())
                else:
                    _logger.error("no access reading file: %s" %
self.filename)
                    result = None
        else:
            if(_data is not None):
                try:
                    result = load_data(_data)
                except Exception, _error:
                    _logger.error(str(_error.message))
            else:
                _logger.error("not find deserialize data!")
                result = None
        return result
    def serialize(self, _data=None):
        result = None
        if(self.filename is not None):
```

```
            if not access(self.filename, F_OK):
                _logger.info("no find file: %s, created new file!" %
self.filename)
                with open(self.filename, "w+"):
                    pass
            if access(self.filename, W_OK):
                with open(self.filename, "w") as config:
                    if(_data is None):
                        _logger.error("not find data!")
                        result = None
                    else:
                        result = dump_data(_data, default_style='"')
                        config.write(result)
            else:
                _logger.error("no access writing to file: %s" %
self.filename)
                result = None
        else:
            if(_data is not None):
                result = dump_data(_data, default_style='"')
            else:
                _logger.error("not find serialize data!")
                result = None
        return result
```

Сериализуем с помощью него исходный объект:

```
from yaml_class import _YAML
_YAML.serialise(order)
```

Пример объекта order в YAML:

```
"lines":
- "carrier": "UPS"
  "cost": !!float "35.0"
  "qty": !!int "1"
  "sku": "BRK3144"
"order_id": "PO453234"
"ship":
  "address": "11345 Colmar Street"
  "city": "Dallas"
  "country": "USA"
  "name": "Camela Yong"
  "phone": !!int "0"
  "state": "NY"
  "zip": !!int "346
```

Как видно, объект в представлении YAML хорошо читаем, даёт полное представление о строении исходного объекта, при этом тип данных сохраняется однозначно.

Созданный нами класс позволяет получить сериализованный объект в виде строки, либо сразу записать объект в файл. В дальнейшим можно использовать данный модуль на стороне Python-программы не только для сериализации объектов, а также YAML-формат можно использовать для хранения конфигураций программы

Простота сериализации/десериализации с использованием YAML, а также хорошая челевекочитаемость сериализованного объекта являются главными преимуществами данного формата.

В конечном счёте, для унификации входных данных компанией Delmar был выбран формат YAML, так как формат хорошо читаем человеком, надежен при использовании внутри языка Python, прост в работе, безопасен.

Литература

1. Иан Грэхем. Объектно-ориентированные методы. Принципы и практика = Object-Oriented Methods: Principles & Practice. – 3-е изд. – М.: «Вильямс», 2004.
2. Гради Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на С++. 2-е изд. / Пер. с англ. – М.:"Издательство Бином", СПб:"Невский диалект", 1998 г.
3. http://lib.custis.ru/YAML
4. http://www.seobuilding.ru/wiki/Python
5. http://www.intuit.ru/studies/courses/71/71/info

УДК 004.413

АВТОМАТИЗИРОВАННАЯ СИСТЕМА СБОРА ДАННЫХ ФИЗИЧЕСКОГО ЭКСПЕРИМЕНТА

Хоченков Алексей Евгеньевич, студент, Ивановский государственный химико-технологический университет, Россия, Иваново, alexkhochenkov@gmail.com
Галиаскаров Эдуард Геннадьевич, к.х.н., доцент, доцент, Ивановский государственный химико-технологический университет, Россия, Иваново, galiaskarov@isuct.ru

Введение

В современном мире информационные технологии вошли практически во все сферы жизни, и наука не является в этом смысле чем-то из ряда вон выходящим, тем более что с развитием научного знания происходит постоянное увеличение сложности задач. В разрешении этой сложности не последнюю роль играет компьютеризация и автоматизация научного эксперимента.

Выполнение любого физического эксперимента укладывается в следующие этапы:

- разработка методики эксперимента;
- сбор данных, запись их на носители, первичная обработка данных;
- вторичная обработка данных, включающая математическую обработку полученных результатов и оценку погрешности;
- анализ результатов.

Автоматизация сбора данных, первичная и вторичная их обработки задачи наиболее простые, а, главное, что именно здесь кроется большая часть рутины в работе экспериментатора. Об автоматизации этих этапов и пойдет речь далее.

Только представьте себе, что в XXI веке исследователь получает результат эксперимента в виде графика на ленте, постоянно делает на ней пометки о значениях диапазона измерения, показаниях вольтметров, численных значениях выдаваемой мощности нагревательных элементов. Ну а после садится за обработку полученных значений, вычитает из них фон прибора [1], рассчитывает термодинамические функции и константы. Сравнимо с набором текста не при помощи современных текстовых редакторов, а на пишущей машинке; кому-то может быть и нравится, но не практично.

Однако так уж получается, что экспериментатор (применительно к масс-спектрометрии) – это в первую очередь физик или химик, то есть ученый, а не специалист в области информационных технологий. Даже наличие цифровых выходов на приборах для исследователя – это замечательный факт, который в теории может быть очень полезен, но освоить эту полезность на практике не представляется возможным. Так происходит потому, что ученые – это те, кто в первую очередь должен разбираться в самих процессах предметной области, а не в микроэлектронике или программировании. Кроме того, современные средства измерения, стали мультифункциональными, что, с одной стороны, делает их более гибкими, а с другой – усложняет общение с ними.

Преодоление этой сложности должно ложиться на плечи специалистов ИТ – системных интеграторов, программистов, системных администраторов и архитекторов, при этом наука как отрасль испытывает острый дефицит специалистов подобного рода.

Собственно, отсутствие в науке программистов, обладающих современными, актуальными знаниями, способных упростить жизнь светлых голов, разрабатывающих действительно хорошие методики экспериментов – это исторически сложившаяся несправедливость, вызванная постоянной нехваткой бюджета. Мастера на все руки, разбирающиеся и в «железе», и в методике эксперимента, и в проектировании современных информационных систем – товар штучный.

Но в условиях постоянного роста научного знания и материалов экспериментов, с ним связанных, игнорирование возможности регистрации цифрового сигнала с последующей его обработкой становится нереальным. Приходится обходиться энтузиастами, которые самостоятельно постигают азы программирования и пишут приложения, поддержка и расширение коих либо вообще невозможна, либо требует затрат колоссального количества трудовых ресурсов и не известно, что лучше: вносить изменение в уже имеющийся код или переписать всю систему заново. Кроме того, базовые знания не позволяют грамотно расходовать вычислительные ресурсы компьютеров, что, в свою очередь, не позволяет экономить и машинное время, и, как следствие, деньги, выделяемые на приобретение новых компьютеров.

Что такое масс-спектрометрия?

Масс-спектрометрия – метод определения химического, фазового состава и молекулярной структуры вещества, основанный на регистрации спектра масс ионов, образованных в результате ионизации атомов и (или) молекул пробы [2].

Масс-спектрометр – прибор, разделяющий с помощью электрических и магнитных полей пучки заряженных частиц (обычно ионов) с разным отношением массы частицы к её заряду [3].

Анализ вещества методом масс-спектрометрии упрощенно можно представить так: в эффузионную камеру загружается исследуемое вещество, которое нагревается вместе с камерой до определённой температуры [4]. При этом регистрируется поток вещества, истекающий из эффузионного отверстия. Первоначально проводится оценка зависимости интенсивности от массы и поиска т.н. пиков – максимального отклонения интенсивности от нулевого значения при константном значении температуры. Выбирается и фиксируется значение массы. Затем происходит анализ зависимости интенсивности от температуры, при выбранном значении массы. На основе этих зависимостей и делается вывод о качественном и количественном составе вещества. Экспериментальные данные представляют собой массивы зависимостей, получаемые с измерителей.

Системный интегратор, глядя на это описание эксперимента, выделяет для себя сразу несколько основных положений. Во-первых, необходимо регистрировать значение интенсивности. Для этих целей был выбран цифровой пикоамперметр американского происхождения, имеющий возможность подключения к ЭВМ через последовательный порт. Кроме того, пикоамперметр имеет аналоговый выход, чтение с которого может производиться быстрее чтения через com-порт. Во-вторых, это наличие показаний отношений массы к заряду, которые в нашем случае измеряются при помощи аналогового датчика Холла. В-третьих, поскольку речь идет о термодинамическом исследовании в каждый момент времени, для масс-спектра необходимо регистрировать значения температуры. Простейший способ – использование термопары. Второй и третий пункты говорят нам об отсутствии цифрового сигнала, а, следовательно, необходимо применить аналогово-цифровой преобразователь, который и был закуплен и представлял собой плату на шине PCI. Приблизительная схема устройств и их соединений показана на рис. 1.

Проблемы и их решение

Большая часть оборудования, с которым мы имели дело в нашей лаборатории, было несерийным и кустарно модифицировано. Именно поэтому и возникала потребность в системной интеграции – разработке решений по автоматизации технологических процессов.

Интеграция измерительных устройств с конкретными источниками сигнала заставляет решать множество вопросов: подбор схемы подключения, экранирование, борьба с помехами. Кроме этого, требуется написать такое программное обеспечение, чтобы

– в случае расширения или изменения каких-либо частей аппаратного обеспечения ПО не пришлось бы переписывать заново, и

– такое ПО полностью удовлетворяло бы пользователя: считывало параметры с первичных преобразователей, создавало бы базу эксперимента и позволяло бы в режиме реального времени не только контролировать ход эксперимента, но и давало бы возможность управления им.

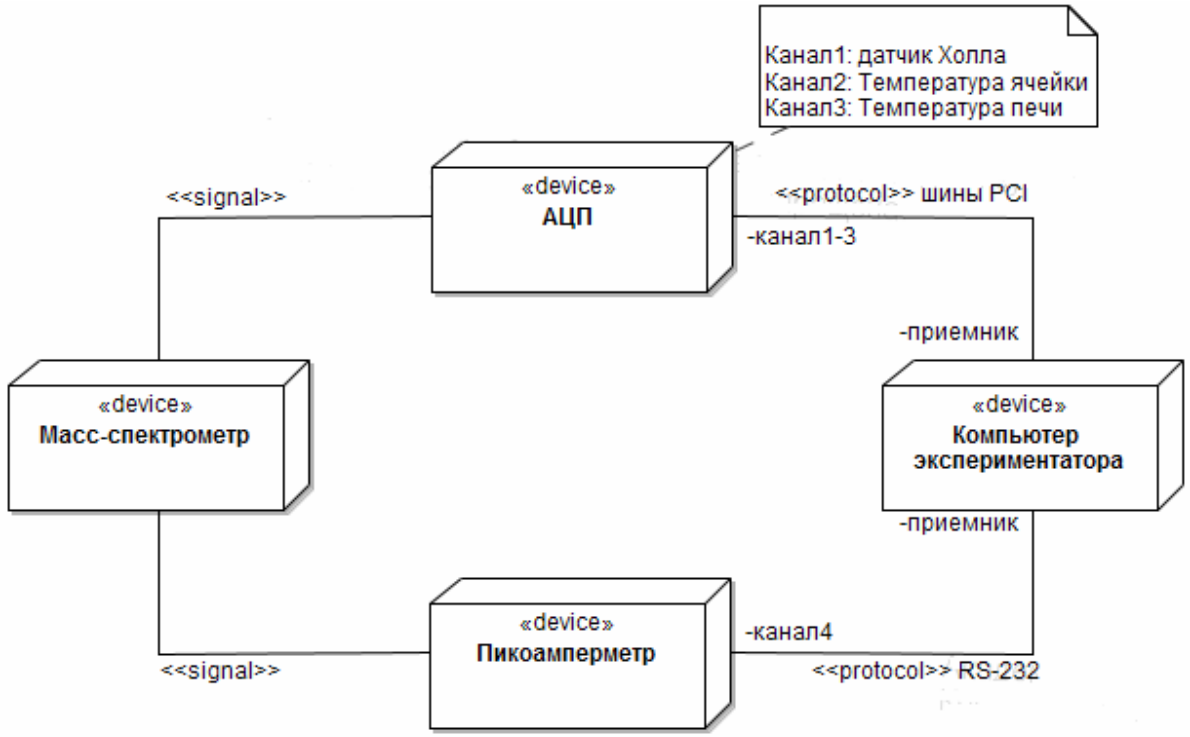

Рис. 1 – Схема подключаемых приборов

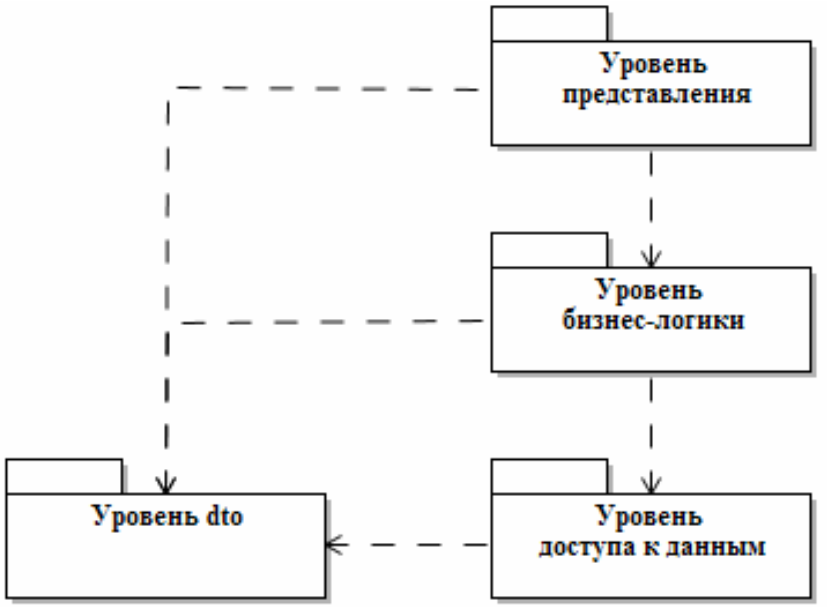

Рис. 2 – "Уровни" приложения

Поэтому за основу была взята многослойная архитектура (рис. 2), подобная архитектурному решению Model-View-Controller, с изменениями, наложенными средой разработки.

В этой архитектуре уровень модели мы называем уровнем доступа к данным. На нем происходит физическая связь с устройствами внутри нашего программного обеспечения.

Уровень представления определяет, как будет выглядеть набор графических форм, а также включает в себя логику их представления.

Уровень контроллера представляет собой так называемый «толстый контроллер», который мы именуем «уровень бизнес-логики». На нем располагаются простые фабрики для объектов и фабрика процессов: получения данных, управления приборами, инициализации доступа к БД и так далее.

К примеру, рассмотрим работу уровней нашей архитектуры при взаимодействии с измерителями. Получится, что уровень доступа к данным знает, КАК отправить сообщение, чтобы получить ответ, а уровень бизнес-логики знает, КАКИМ именно должно быть содержание у этого сообщения. Последний слой в иерархии не содержит никакой логики, а просто хранит структуру тех объектов-сущностей, которыми мы манипулируем в программе; уровень dto (data transfer object) знает, КАКИМИ сущностями оперирует приложение внутри себя. При этом сущности data transfer object создаются при помощи простых фабрик, что позволяет легко понять с какого измерителя к нам «приходит» то или иное значение свойства.

Такая простая иерархия позволяет легко расширять приложение для работы с новыми устройствами.

Если рассмотреть случай с пикоамперметром, принимающим данные с одного канала, и мультиконвертером, обслуживающим другие три, на уровне бизнес-логики, то его можно проиллюстрировать следующей диаграммой классов (рис. 3):

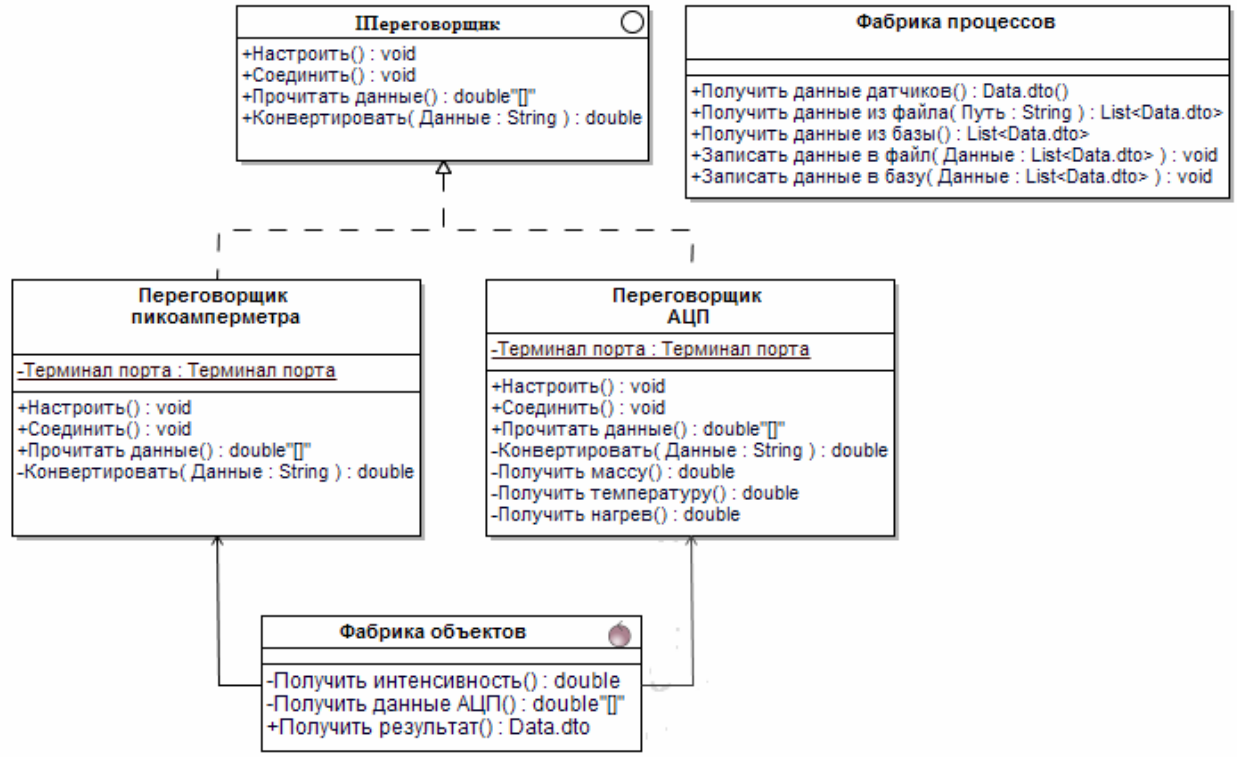

Рис. 3 – Некоторые классы уровня бизнес-логики

Уровень бизнес-логики содержит в себе два класса-посредника, осуществляющих общение этого уровня с использующимися измерителями через слой логики доступа к данным. К этим классам относятся «Переговорщик пикоамперметра», используемый для общения с пикоамперметром, и класс «Переговорщик мультиконвертера», общающийся с мультиконвертером. При этом процессы получения данных с устройств должны производиться в параллельных потоках. Класс «Фабрика объектов» проводит компоновку объекта – результата измерения, управляя переговорщиками и получая от них полезные

данные. А класс «Фабрика процессов» управляет (запускает или останавливает) процессами бизнес-логики.

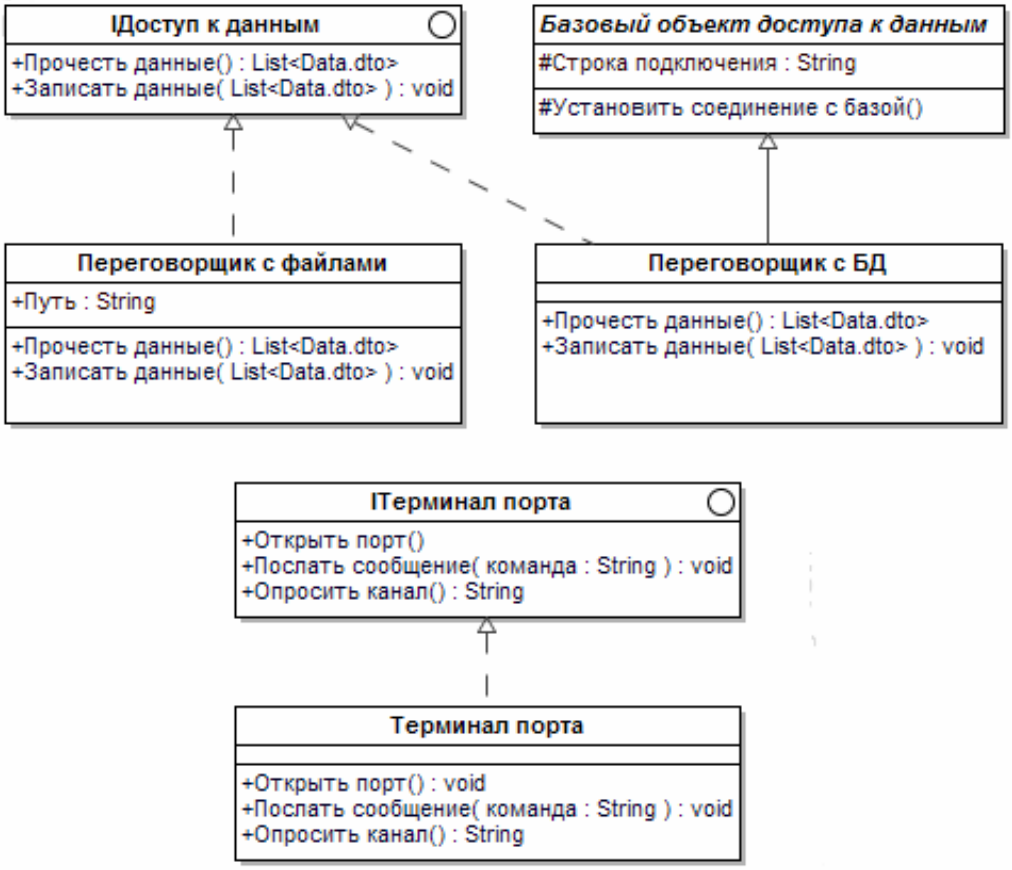

Рис. 4 – Классы уровня доступа к данным

Уровень доступа к данным содержит классы (рис. 4), общающиеся непосредственно с интерфейсами системы – портами, получающими значения с первичных преобразователей, файловой системой и СУБД. Терминалы портов посылают по предварительно открытым каналам предопределенные переговорщиками пикоамперметра и мультиконвертера сообщения и получают ответ от порта, никак его при этом не изменяя. А переговорщики этого уровня с файлами и БД выполняют простейшие операции записи/чтения из файла при помощи стандартных классов пространства имен System.IO.FileStream и выполняют простейшие SQL-запросы к базе данных на чтение и запись массива данных.

Взаимодействие уровней бизнес-логики и доступа к данным можно увидеть на диаграмме последовательностей (рис. 5). Опрос портов переговорщиками при этом происходит в параллельных потоках. Результом работы фабрики объектов становится объект Data.dto, который передается фабрикой процессов уровню представления системы (передача данных уровню представления на диаграмме последовательности опущена).

Выводы

Таким образом, предложена архитектура системы для сбора и обработки данных, получаемых в ходе физического эксперимента, в которую достаточно легко вносить изменения в случае модернизации или смены оборудования, за счет применения интерфейсов, порождающего паттерна «Фабрика объектов» и структурного паттерна «Фасад». Кроме того, использование такого подхода позволяет решать проблемы сложности внутри программного обеспечения. Каждый из уровней представляет собой отдельную подсистему (это значительно упрощает внесение изменений в систему), предоставляющую другим подсистемам интерфейсы для взаимодействия; все классы верхнего уровня используют нижележайший слой через интерфейс, избавляя тем самым приложение от лишних зависимостей.

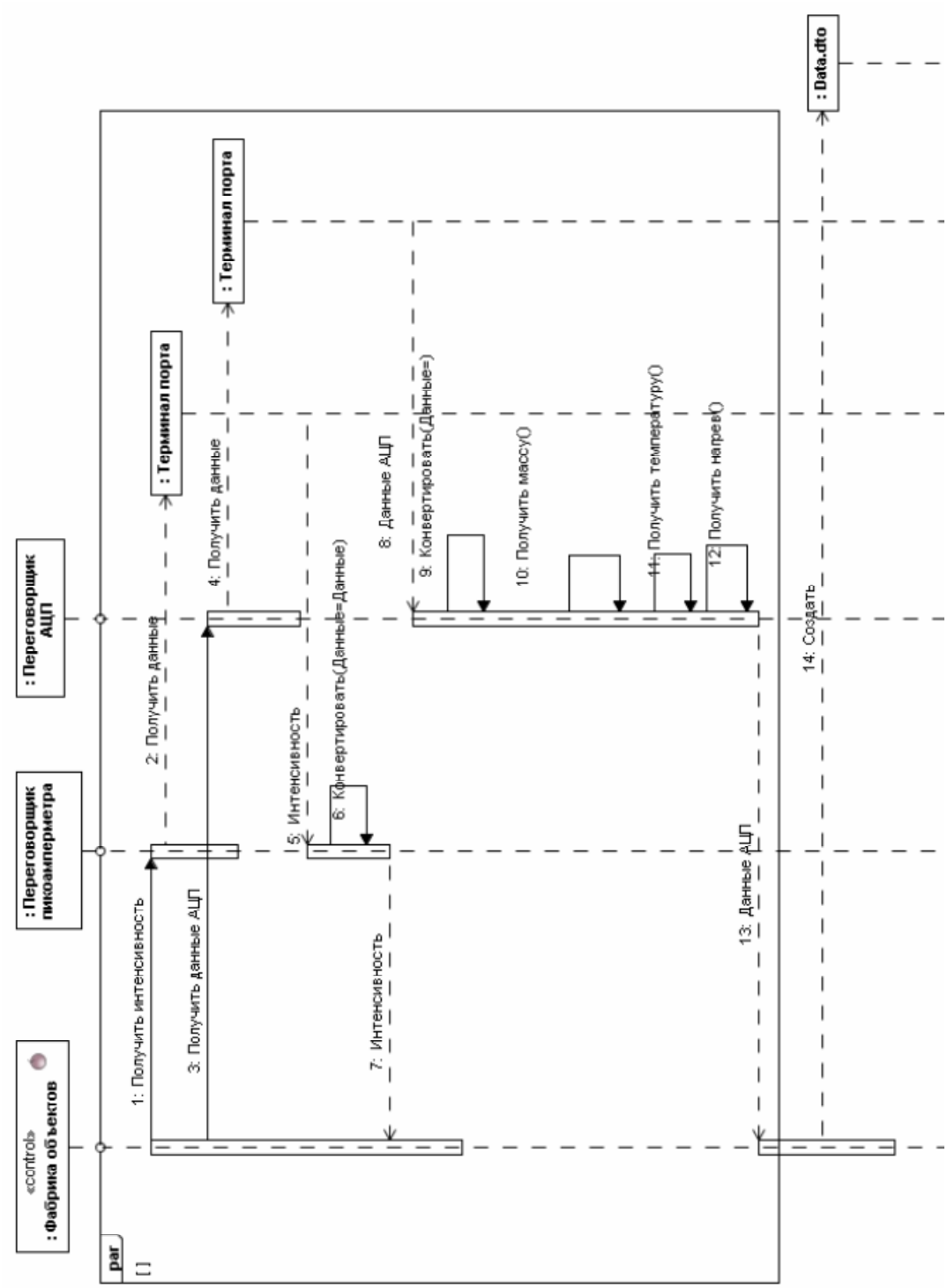

Рис. 5 – Диаграмма последовательности обмена сообщениями между объектами приложения при чтении данных в реальном времени

Литература

1. Зенкевич И.Г., Иоффе Б.В. Интерпретация масс-спектров органических соединений. Л.: Химия, 1986. 176 с.
2. Г.И. Рамендик. Масс-спектрометрия // Горная энциклопедия. Т.3, 1987 – с. 266 – 267
3. А.В. Дубровин, Г.И. Косоуров. Масс-спектрометр // Физический энциклопедический словарь. Т.3, 1963 – с. 138-146
4. Сидоров, Л. Н. Масс-спектральные термодинамические исследования / Л. Н. Сидоров, М. В. Коробов, Л. В. Журавлева. – М.: Изд-во Моск. ун-та, – 1985 г. – 208с.

УДК 004.41

ТЕХНОЛОГИЯ РАЗРАБОТКИ СИСТЕМЫ АВТОМАТИЗАЦИИ ПРОЕКТИРОВАНИЯ «NETWORK LAB»

Прилепко Максим Анатольевич, Омский государственный технический университет, Россия, Омск, Zeitgeist_89@mail.ru

В технических ВУЗах получение студентами практических навыков в области информационно-коммуникационных систем иногда затруднено тем, что студент не имеет возможности отрабатывать те или иные практические задания в связи с отсутствием оборудования для построения локальных вычислительных сетей.

Решением данной проблемы является использование в учебном процессе виртуальных тренажеров. Однако систем для создания таких тренажеров в области сетевых технологий в настоящее время мало, многие из них специализируются на изучении сетевого оборудования определенной компании и являются дорогостоящими. Поэтому разработка систем автоматизации проектирования виртуальных тренажеров по изучению общих принципов создания сетей и общих принципов работы сетевого оборудования является актуальной задачей и имеет практическую значимость для использования современных технологий при подготовке специалистов, особенно при дистанционной форме обучения.

В качестве языка проектирования системы автоматизации «Network Lab» (рис. 1) был выбран язык UML [1]. Выбор данного языка был сделан исходя из таких преимуществ:

- универсальности языка UML;
- методы описания результатов анализа и проектирования на языке UML семантически близки к методам программирования на современных объектно-ориентированных языках;
- позволяет описать разные аспекты поведения системы.

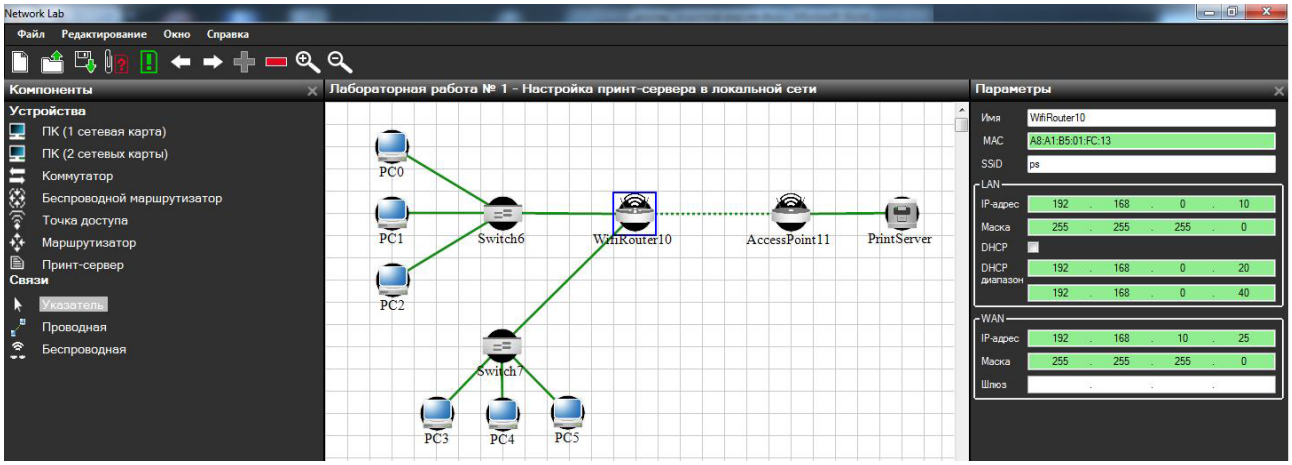

Рис. 1- Пример проекта тренажера «Network Lab» в режиме создания

При создании «Network Lab» язык UML применялся на следующих стадиях реализации проекта [2-4]:

1. *Анализ требований* – цель данной стадии состоит в том, чтобы понять процессы, которые управляют предприятием или системой, определить область деятельности системы и требования пользователя [5]. На данной стадии с помощью UML создавалась модель прецедентов системы. Она позволила выделить внешние системы, контактирующие с системой, основные процессы и их взаимосвязь. Диаграммы прецедентов дали возможность выделить функциональную структуру системы, не вдаваясь в детали ее реализации. Кроме того, было произведено предварительное выделение объектов системы и их классификация. На основании построенной модели был составлен план разработки системы.

2. *Системное проектирование* – данная стадия включает в себя решения верхнего уровня относительно разработки системы в целом. Здесь производилась разработка архитектуры системы с помощью диаграммы развертывания. Она позволила выделить вычислительные ресурсы, устройства, использующиеся ими, и соединения между ними, спроектировать размещение отдельных процессов и исполняемых компонент на определенных устройствах, что особенно важно при проектировании сложных систем и интернет-приложений [5].
3. *Детальное проектирование* – на данной стадии описывались способы решения задач, выполняемых системой. Эта стадия полностью описывает функции, классы системы и графический интерфейс с пользователем. На данной стадии были разработаны диаграммы классов, включая отношения между классами и их атрибутами, что позволило произвести классификацию объектов, функционирующих в системе. С применением диаграммы поведения (диаграммы последовательности, диаграммы взаимодействия, диаграммы состояний и диаграммы активности) была разработана модель поведения объектов в системе.

С помощью разработанной модели поведения были установлены зависимости между классами, произведено разделение системы на модули и выделены классы, реализуемые в данных модулях.

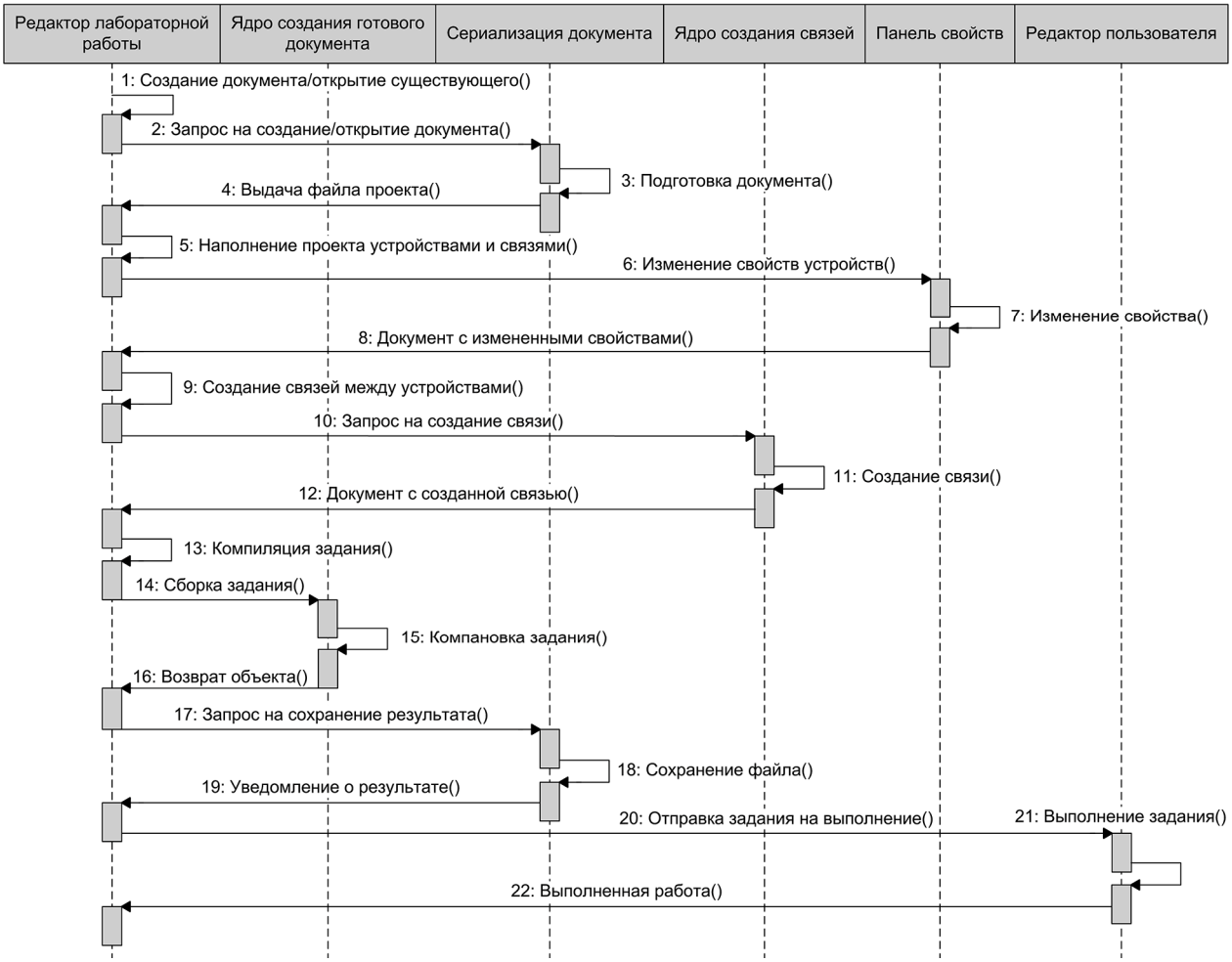

Рис. 2 – Диаграмма последовательности процесса создания виртуального тренажера в нотации UM

Система автоматизации проектирования «Network Lab» включает в себя два исполняемых компонента (подсистемы):

1. подсистема создания виртуальных тренажеров;
2. подсистема воспроизведения виртуальных тренажеров.

Редактор лабораторной работы отвечает за наполнение лабораторной работы сведениями (теоретическим материалом) в ходе диалога с пользователем, а также за управление действиями и инструментами. Ядро создания готового документа отвечает за сборку и компиляцию виртуальной лабораторной работы. Сериализация документа занимается процессами подготовки документа при создании и сохранении файла виртуального тренажера.

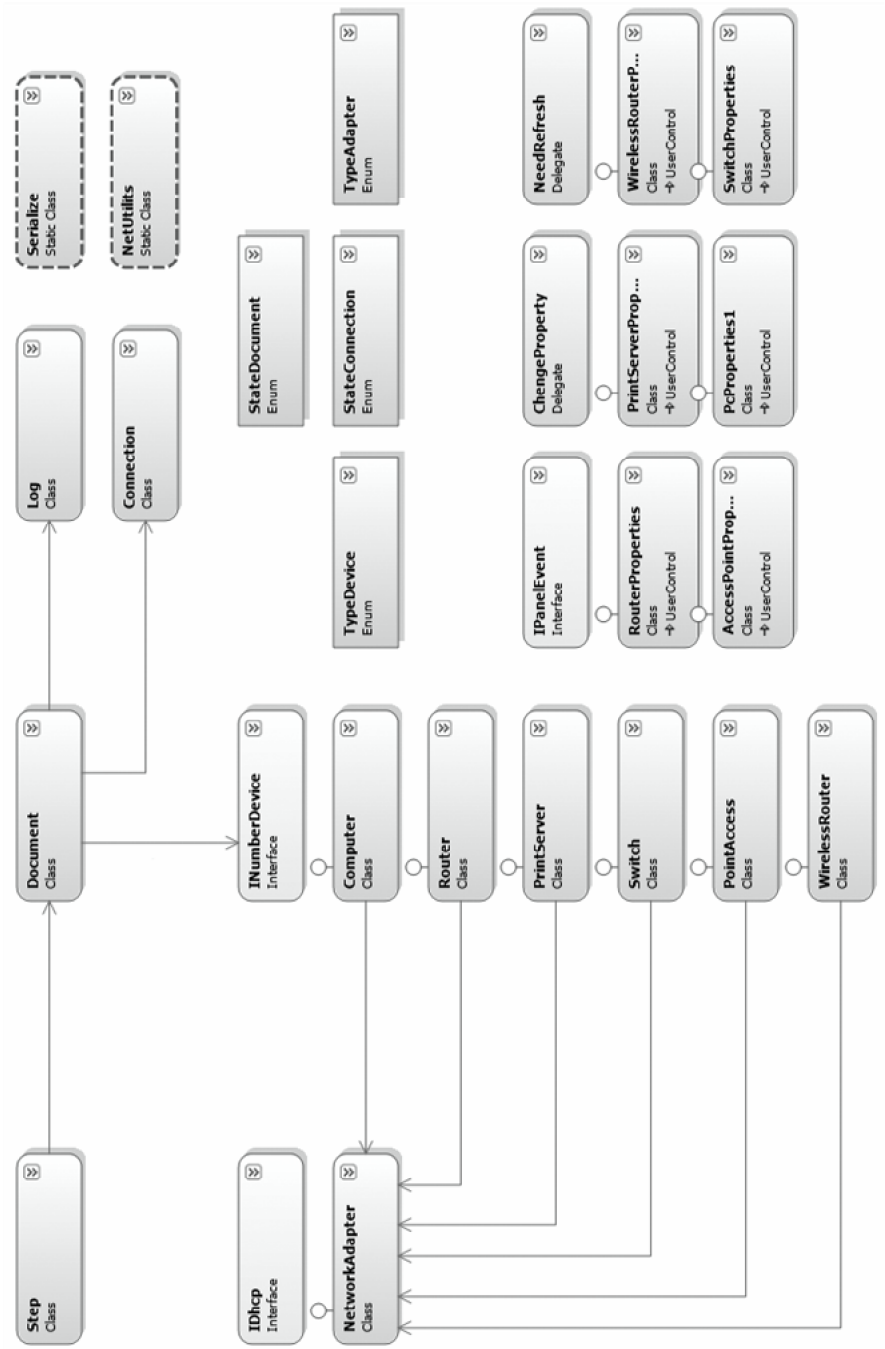

Рис. 3 -Диаграмма классов в нотации UML для системы автоматизации проектирования «Network Lab»

Ядро создания связей отвечает за проверку доступности компонентов, а также отображение разных видов соединения. Панель свойств отвечает за изменение настроек компонентов виртуального тренажера. Редактор пользователя занимается процессом выполнения виртуального тренажера.

Общий ход взаимодействия компонентов системы в ходе создания тренажёра можно изобразить в виде диаграммы последовательности в нотации UML [1], представленной на

рис. 2.

Подсистема создания виртуального тренажера выполняет центральную роль, координируя весь процесс построения виртуальной лабораторной работы. Так как главной функцией редактора виртуальных тренажёров является создание экземпляров сущностей, описанных в модели данных, их редактирование и хранение, им должны выполняться следующие задачи:

1. создание и редактирование общей информации о лабораторной работе;
2. создание и редактирование компонентов лабораторной работы;
3. создание и редактирование связей между компонентами;
4. создание и редактирование параметров компонентов.

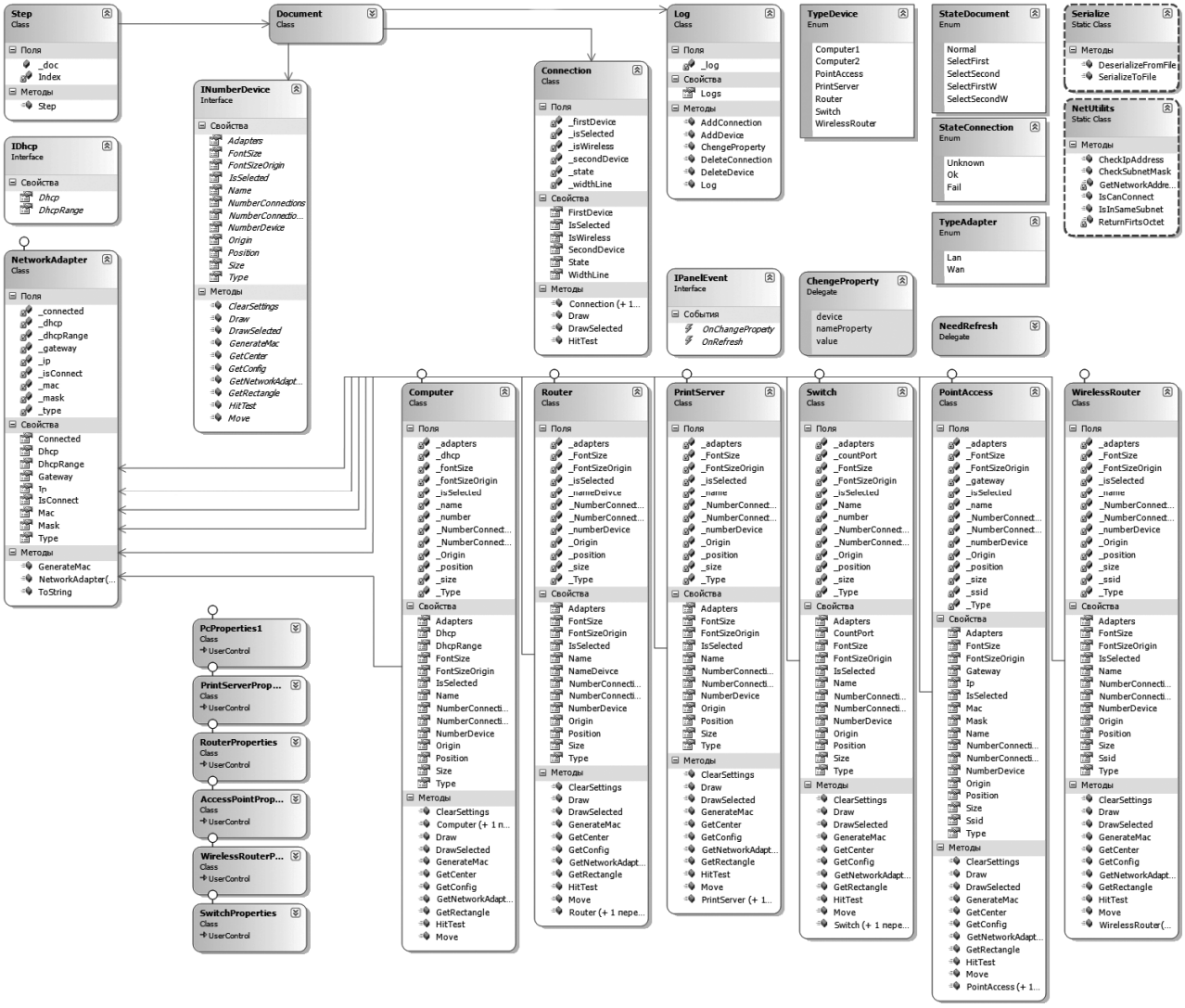

Рис. 4. Развернутая диаграмма классов системы автоматизации проектирования «Network Lab» в нотации UML

Главная функция подсистемы воспроизведения виртуальных тренажеров – отображение учебного материала и сбор сведений о процессе выполнения виртуальной лабораторной работы пользователем. Подсистема воспроизведения выполняет следующие функции:

1. загрузка структуры лабораторной работы из файла;
2. чтение ресурсов лабораторной работы (текста и изображений);
3. исполнение сценария лабораторной работы;
4. сбор и сохранение сведений о выполнении лабораторной работы пользователем.

Основу архитектуры системы автоматизации проектирования «Network Lab»

составляет класс Step, в котором содержится класс Document, отвечающий за структуру проекта виртуального тренажера. Каждый компонент оборудования тренажера имеет свой класс и присущий ему набор свойств и методов. За ведение истории действий пользователя отвечает класс Log. Также в системе описаны классы Serialize (для сериализации документов) и NetUtilits (для проверки доступности узлов и реализации команды ping) (рис. 3, 4).

Визуальные модели все более широко используются при проектировании систем, сложность, масштабы и функциональность которых постоянно возрастают. Они хорошо приспособлены для решения таких часто возникающих при создании систем задач, как физическое перераспределение вычислений и данных, обеспечение параллелизма вычислений, обеспечение безопасности доступа к ИС, оптимизация балансировки нагрузки ИС, устойчивость к сбоям и т.п.

В зависимости от сложности предметной области, квалификации проектировщика и качества вспомогательных программных средств, процесс проектирования информационных систем на основе UML может значительно ускорить и упростить процесс проектирования. Этап диаграммного моделирования обеспечивает такие преимущества, как обнаружение и исправление логических неточностей на раннем этапе проектирования. Созданные диаграммы в дальнейшем составляют часть документации программного продукта. Визуализированные средства UML модели ИС обеспечивают ясность представления выбранных архитектурных решений и позволяют понять разрабатываемую систему во всей ее полноте.

Литература

1. Фаулер, М. UML. Основы: Краткое руководство по стандартному языку объектного моделирования: пер. с англ. / М. Фаулер. – 3-е изд. – СПб.: Символ–Плюс, 2004. – 192 с.: ил.
2. ГОСТ 34.003-90. Автоматизированные системы. Термины и определения. – Взамен ГОСТ 24.003-84, ГОСТ 22487-77; введ. 1992-01-01. –М.: Изд–во стандартов, 1991.– 21 с.
3. ГОСТ 34.601-90. Автоматизированные системы. Стадии создания. – Введ. 1992-01-01. – М.: Издательство стандартов, 1991. – 8 с.
4. Информационная технология. Комплекс стандартов и руководящих документов на автоматизированные системы: ГОСТ 34.201-89, ГОСТ 34.602-89, РД 50-682-89. – Введ. 1990-01-01. – М.: Издательство стандартов, 1989. – 32 с.
5. Мищук, А, Применение UML в жизненном цикле проектов [Электронный ресурс] / А. Мищук. – 2009. – Режим доступа: http://www.interface.ru/fset.asp?Url=/rational/umlob.htm

УДК 541.662:04.004

ПРОЕКТИРОВАНИЕ ЯДРА ИНФОРМАЦИОННОЙ СИСТЕМЫ ЭКОНОМИЧЕСКОГО ПРОИЗВОДСТВЕННО-ЭНЕРГЕТИЧЕСКОГО КЛАСТЕРА УГОЛЬНОЙ ПРОМЫШЛЕННОСТИ

Кураков Юрий Иванович, д.т.н., профессор, заведующий кафедрой "Естественнонаучные дисциплины", ШИ (ф) ЮРГПУ (НПИ) им. М.И. Платова, phisycs@yandex.ru
Олейник Павел Петрович, к.т.н, системный архитектор программного обеспечения, ОАО "Астон", доцент кафедры "Естественнонаучные дисциплины", ШИ (ф) ЮРГПУ (НПИ) им. М.И. Платова, xsl@list.ru

Каждая организация или группа компаний в настоящее время использует единую корпоративную информационную систему для удовлетворения информационных потребностей.

В данной статье рассматривается вопрос проектирования ядра информационной системы, автоматизирующей деятельность группы предприятий угольной промышленности, организованных в экономический кластер с целью повышения общей рентабельности и

снижения себестоимости конечного продукта. В работе освещена структура экономического кластера, описана необходимость наличия отдельных звеньев и определена номенклатура выпускаемой продукции, получаемой при глубокой переработке углеродсодержащих ископаемых. В конце статьи представлена UML-диаграмма классов, составляющих ядро информационной системы экономического производственно-энергетического кластера угольной промышленности.

На наш взгляд, экономически целесообразно организовать производственно-технологический кластер по добыче и глубокой переработке угля на основании схемы, представленной на рис. 1. Схема построена на основе многолетних научных исследований, выполненных одним из авторов статьи [1-5]. Объем статьи не позволяет подробно рассмотреть рисунок, поэтому опишем его лишь кратко.

Рис. 1 – Структурный план организации экономического производственно-энергетического кластера

Выданная нагора горная масса кроме угля всегда содержит определенное количество породы. Кроме того, большое количество породы образуется при подготовительных работах (нарезание лав, проходке штреков, бремсбергов и др). Первичная переработка горной массы осуществляется на обогатительной фабрике. Отделённая от угля порода используется для производства аглопорита, сульфата алюминия.

Порода угольных шахт содержит широкий спектр различных химических соединений и элементов. Это германий и редкоземельные элементы, железная руда, глинозём для производства бокситов. Из обогатительной фабрики порода транспортируется в цех по её переработке, который относится к комбинату углеродных изделий.

После обогатительной фабрики и передела на стадии отмыва антрацита от угольной пыли (производство гидроантрацита, т.е. фильтранта) штыб и шлам попадает в цех брикетирования. Брикет угольный представляет собой смесь компонентов: штыб антрацитовый, шлам антрацитовый, шлам каменноугольный и связующие элементы. В качестве связующего элемента можно применить каменноугольный пек, нефтяной пек, некоторые полимеры.

При добыче и обогащении углей образуются отходы – это шламы и тонкие мелкодисперсные и высокозольные продукты, содержащие от 30 до 80% органической массы, на основе которой изготавливают водоугольное топливо.

Водоугольное топливо (ВУТ) разработано и внедрено в промышленном масштабе в США, Италии, Японии, Германии, Польше, Китае, Швеции и других государствах в 60-е годы прошлого столетия.

Водоугольные суспензии являются одной из оптимальных форм нового вида топлива. В качестве добавок, улучшающих физические свойства суспензий, предлагаются в основном соединения четырех типов: 1 – анионные ПАВ (поверхностно-активное вещество); 2 – неионногенные ПАВ; 3 – сополимеры на основе акриловой кислоты, полиэфирные соединения и др., 4 – щелочные добавки.

Шламообразование на угледобывающих предприятиях происходит за счет того, что шахтным водопритоком мелкие частицы угля из подготовительных и очистных забоев, из транспортных выработок выносятся в водоотливные камеры и далее с шахтной водой выдаются в отстойники. На обогатительных фабриках шламообразование происходит при некачественных процессах обогащения, осветления, классификации и обезвоживании мелких классов угля (- 0,5 мм составляет до 90%). Выход шламов достигает 10% от общего объёма перерабатываемого угля.

Низкозольные шламы можно использовать как сырьё для шихтования или компонента концентрата, как сырьё для изготовления брикетов или получения ВУТ. Цена ВУТ в 2-4 раза меньше цены мазута.

Примером топливно-энергетического кластера с технологией глубокой переработки антрацита могло бы послужить предприятие, образованное путём объединения таких организаций, как ЗАО «Сибирский антрацит» (которое в настоящее время занимается лишь добычей антрацита) и завода НовЭЗ, занимающегося глубокой переработкой углеродсодержащих материалов.

Перейдем к рассмотрению реализации программного обеспечения для экономического кластера. В настоящее время наибольшую популярность получила разработка в соответствии с принципами MDA (Model Driven Architecture), предполагающими наличие развитой метамодели объектной системы, интерпретируемой при выполнении приложения. Реализация метамодели, используемая при реализации ядра кластера, описана в работах [6-7] и представлена в виде диаграммы классов языка UML на рисунке 2.

Базовым классом рассматриваемой иерархии является MetaModelItem, который содержит общие свойства и методы, характерные для всех элементов метамодели. Основное свойство описываемого класса содержит тип самого элемента и используется многократно для получения метаинформации. Элементы метамодели, которые должны иметь название и заголовок (класс, атрибут класса, ассоциация и т.п.) унаследованы от CaptionedMetaModelItem. Для представления различных классов системы используется корневой абстрактный класс Class.

В системе предусмотрено два вида наследования. Первое предполагает простое наследование классов без возможности добавления атрибутов пользователем. Данный тип наследования может использоваться классами лингвистического транслятора, который предполагает создание иерархии классов, представляющей элементы русского языка (например, морфемы). Для решения описанной задачи введён абстрактный класс SimpleInheritanceClass.

Для реализации классического наследования, предполагающего добавление как унаследованных классов, так и определения имеющихся атрибутов, используются производные от CustomAttributedClass классы. В частности реализованный (неабстрактный) DomainClass класс введён для реализации сущностей предметной области, каждая из которых может быть унаследована от нескольких базовых. Множественное наследование позволяет проектировать более гибкие системы, автоматизирующие широкий спектр прикладных предметных областей. Например, при проектировании классов, представляющих некоторые товары, может потребоваться реализовать их структуру в виде

дерева. Т.е. необходимо одновременно наследоваться от базового класса представляющего товар, и от системного класса, представляющего древовидную структуру.

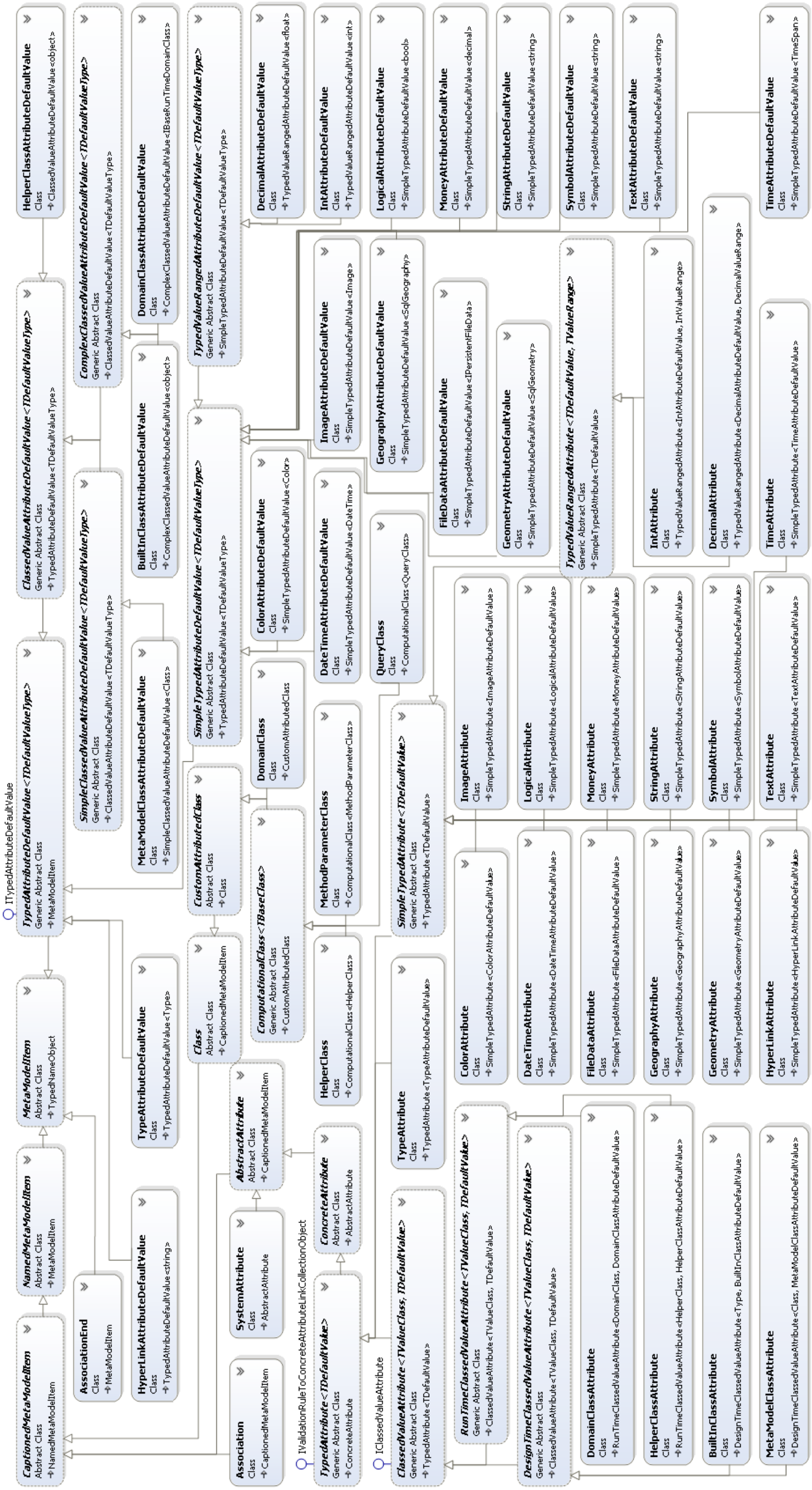

Рис. 2 – Метамодель объектной системы

При проектировании иерархии атомарных литеральных типов (целых чисел, строк, дробных чисел и т.п.) используется общий подход, подробно описанный в работе [7]. Однако с целью упрощения процесса рефакторинга было принято решение о сокращении набора классов и применение параметризованных типов (Generic в языке C#). Корневым является класс AbstractAttribute, от которого унаследованы SystemAttribute и ConcreteAttribute. Первый используется для описания атрибутов системных классов, которые не могут редактироваться или добавляться пользователем. Второй представляет непосредственно атрибуты, добавляемые пользователем в режиме выполнения приложения.

Базовый параметризованный класс TypedAttribute является корневым для всех атрибутов, у которых параметр типа выступает в качестве типа данных для значения атрибута. Абстрактный класс SimpleTypedAttribute выступает корнем иерархии атомарных литеральных типов. В свою очередь ClassedValueAttribute выступает корнем для всех атрибутов, значениями которых выступает объект определенного класса.

Для представления связей между классами используется механизм ассоциаций языка UML [7], для реализации которого выделен класс Association. В системе предусмотрена возможность описания только бинарных связей, т.к. для них проще всего определить кратность каждого участника. Поэтому для представления отдельного края ассоциации используется экземпляр класса AssociationEnd.

С глобальной точки зрения, представленная метамодель позволяет описать любую предметную область, в том числе и экономический кластер. На рис. 3 представлена иерархия классов экономического угольно-энергетического кластера, для проектирования которой использовалась описанная метамодель.

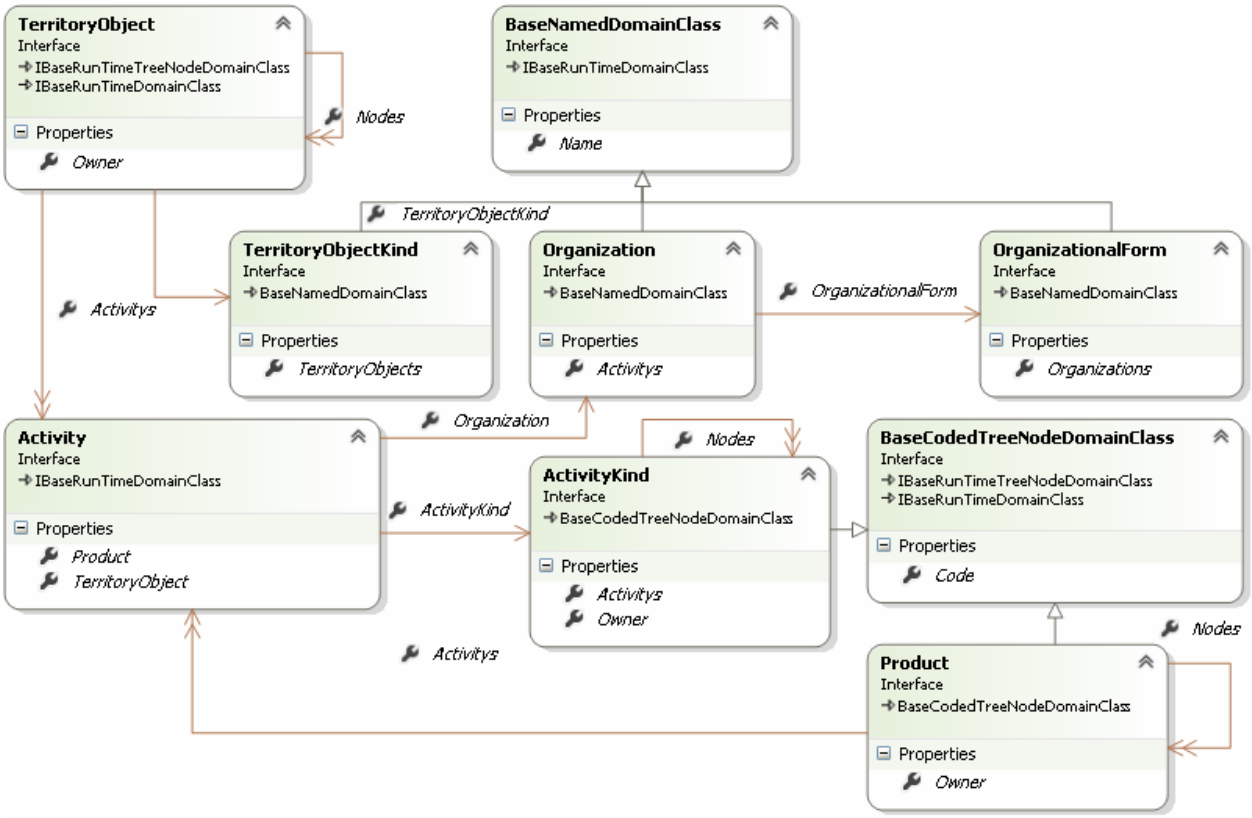

Рис. 3 – Иерархия классов экономического производственно-энергетического кластера

Рассмотрим изображённую иерархию более подробно. Для представления территориальных объектов создан класс TerritoryObject. Класс TerritoryObjectKind описывает виды территориальных объектов: страна, округ, город и т.п. Для представления поставщиков/потребителей углеродсодержащих материалов используется класс Organization, а для организационно-правовой формы – класс OrganizationForm. Для сохранения информации о товарах и услугах введён класс Product, структура которого соответствует

Общероссийскому классификатору продукции. Каждая организация одновременно может быть поставщиков одного вида продукции и потребителем другого. Для унифицированной реализации подобной структуры выделено два класса: 1) ActivityKind – для описания видов деятельности (продажа, производство, логистика и т.п.); 2) Activity – для непосредственного описания деятельности (например, указания конкретного антрацита, выработанного на определённой угольной шахте в конкретном населенном пункте). Одной из ключевых особенностей объектно-ориентированного проектирования является наследование, предоставляющее возможность выделения базовых абстрактных классов, содержащих общие свойства, для производных классов. Так, класс BaseNamedDomainClass содержит атрибут Name, содержащий название. Класс BaseCodedTreeNodeDomainClass содержит атрибут Code, используемый для сохранения кодов классификаторов.

Многие из представленных классов (TerritoryObject, Organization, Product, ActivityKind) необходимо организовать в древовидную структуру данных. Именно поэтому рассматриваемые классы унаследованы от системного класса IBaseRunTimeDomainClass, имеющегося в метамодели объектной системы [6-7]. Описанные действия привели к автоматической генерации свойства Nodes, содержащего дочерние (вложенные) узлы. Все выделенные ассоциации являются бинарными и двунаправленными. Т.е. каждый класс, участвующий в ассоциации, имеет атрибут, представляющий класс противоположного края ассоциации.

Представленная иерархия классов экономического кластера является базовым ядром разрабатываемой системы. В настоящий момент в систему заносится информация об угольных шахтах Ростовской области и обо всех ключевых потребителях углеродсодержащих материалов. Дальнейшее развитие предполагает миграцию в сторону Веб-приложения и реализацию различных элементов личного кабинета пользователя, а также развитых средств поиска информации по введенным параметрам. Отметим, что представленная иерархия является унифицированной и позволяет автоматизировать деятельность экономического кластера для любой отрасли (а не только угольной).

В статье представлена структура экономического производственно-энергетического кластера глубокой переработки углеродсодержащих материалов. Детально описана номенклатура выпускаемой продукции, рассмотрена структура ядра единой управляющей информационной системы для автоматизации деятельности кластера.

Литература

1. Кураков Ю. И. Влияние отвалов угольных шахт на состояние атмосферы// Химия твердого топлива. – 2005. № 6. – С. 70–76.
2. Кураков Ю. И., Передерий М.А., Самофалов В.С. Углеродные молекулярные сита из антрацита// Изв.вузов. Сев.-Кавк. регион. Естеств.науки. Приложение. 2004. №1. – С.84-92.
3. Кураков Ю. И., Передерий М.А., Самофалов В.С. Дробленные и гранулированные сорбенты из антрацита// Химия твердого топлива. – 2004. № 3. – С. 46-59.
4. Кураков Ю. И. Износостойкость композитов на основе антрацитов. Изв. Вузов// Сев.-Кавк. регион . Техн. науки. Спец. выпуск. 2005. – С . 46 – 53.
5. Кураков Ю. И. Применение антрацитов Донбасса в ка-честве наполнителей электродных изделий// Химия твердого топлива. – 2006. № 3. – С. 68-76.
6. Олейник П.П. Иерархия классов метамодели объектной системы // Объектные системы – 2012: материалы VI Международной научно-практической конференции (Ростов-на-Дону, 10-12 мая 2012 г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ ЮРГТУ (НПИ), 2012. – С. 37-40.
7. Олейник П.П. Элементы среды разработки программных комплексов на основе организации метамодели объектной системы // Бизнес-информатика. 2013. №4(26). – С. 69-76.

УДК 681.3

МОДЕЛИРОВАНИЕ УЧЕБНОГО ПРОЦЕССА С ИСПОЛЬЗОВАНИЕМ ОНТОЛОГИИ ЖИЗНЕННОГО ЦИКЛА

Никулин Валентин Валерьевич, ГОУ ВПО «Уральский федеральный университет имени первого Президента России Б.Н.Ельцина», Нижнетагильский технологический институт (фил.). Факультет экономики и менеджмента, кафедра информационных технологий, студент. Россия. Нижний Тагил. dragovicntn@gmail.com

Пахмутов Александр Владимирович, ГОУ ВПО «Уральский федеральный университет имени первого Президента России Б.Н.Ельцина», Нижнетагильский технологический институт (фил.). Факультет экономики и менеджмента, кафедра информационных технологий, студент. Россия. Нижний Тагил. fatalitytnt@gmail.com

Грегер Сергей Эдуардович, ГОУ ВПО «Уральский федеральный университет имени первого Президента России Б.Н.Ельцина», Нижнетагильский технологический институт (фил.). Факультет экономики и менеджмента, кафедра информационных технологий, доцент. Россия. Нижний Тагил. segreger@gmail.com

В процессе обучения все люди в разном темпе усваивают теоретический материал, подкрепленный простыми и сложными задачами. Некоторые понимают “на лету”, другим же требуются дополнительные усилия. В большинстве случаев такие усилия представляют собой применение теоретических знаний на практике. Без “наставника” это довольно трудоемкая работа.

Сейчас в Internet огромное количество ресурсов, предназначенных для подобной самоподготовки. В основном эти ресурсы представляют собой архивы заданий с онлайн-системой проверки результата или же огромные теоретические справочники, книги в web-представлении. Вышеупомянутые архивы заданий имеют один существенный недостаток. С их помощью пользователь нерационально изучает интересующий его материал, ему приходится самостоятельно искать нужные данные в книгах и на страницах интернета. Пользователь может не знать идею решения, а также о существовании альтернативных способов получения правильного ответа**,** и**,** помимо этого, не получает никаких советов в процессе самого решения задания.

В данной статье под учебным процессом подразумевается процесс самообучения с помощью поэтапного решения практических заданий, в ходе которого пользователь получает теоретические советы, рекомендации, формулы.

Цель работы – создать систему, предоставляющую пользователю банк задач (заданий), с дополнительными функциональными возможностями, которые будут решать следующие задачи:

- предоставление возможных алгоритмов решения;
- поэтапное сопровождение по алгоритму решения, с возможностью проверки промежуточных результатов;
- предоставление дополнительных статистических оценок сложности алгоритмов;
- В ходе работы были выделены следующие основные классы объектов (Рисунок 1):
- дисциплина;
- тема;
- задача (как основная информационная единица системы);
- алгоритм;
- стадия алгоритма.

Так же, перед реализацией системы необходимо разделить роли пользователей. Проиллюстрируем данное деление с помощью диаграммы прецедентов (Рисунок 2).

С помощью дополнительного модуля CMS Plone OntoEditor, созданного сотрудниками и студентами нашей кафедры, была спроектирована и реализована онтология.

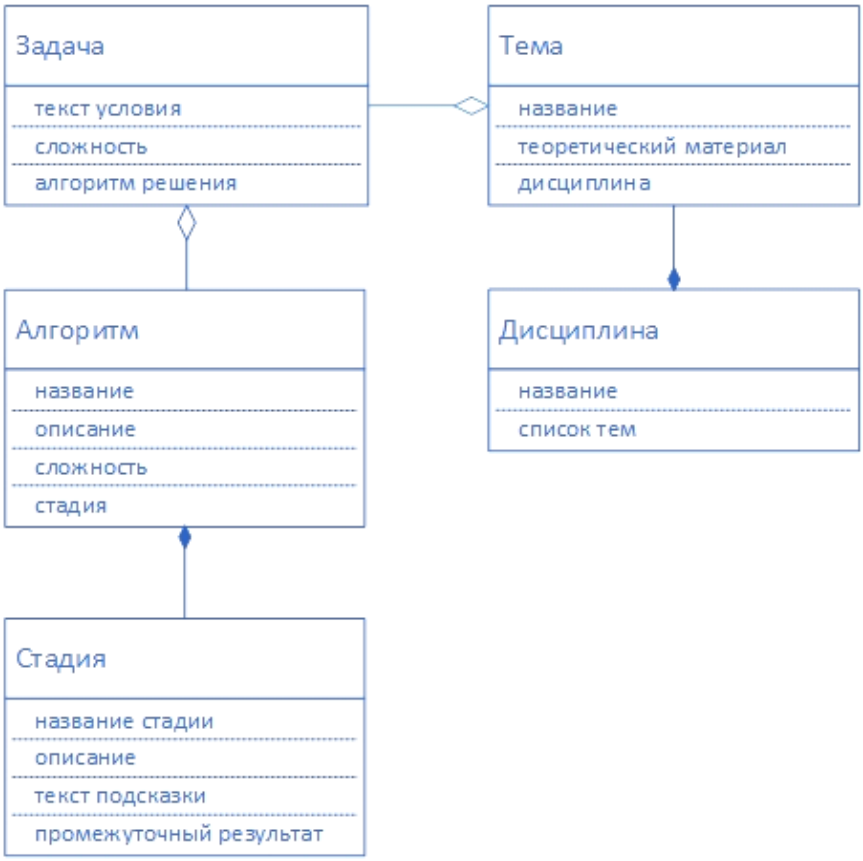

Рис. 1 – UML диаграмма классов

Данный модуль позволяет в интерактивном, удаленном и визуальном режиме создать онтологию и наполнить ее знаниями, т.е. создать семантическую сеть. Она представляет собой ориентированный граф, вершины которого соответствуют объектам предметной области, а дуги задают отношения между ними. Представление онтологии в окне редактора показано на рисунке (Рисунок 3).

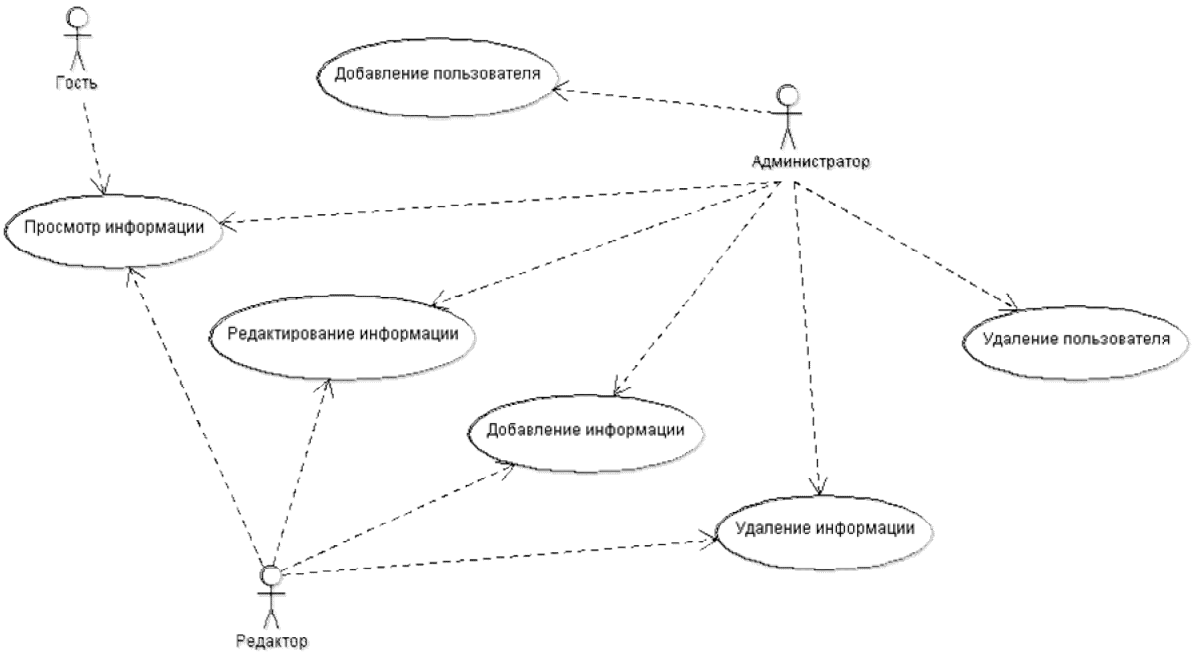

Рис. 2 – Диаграмма прецедентов

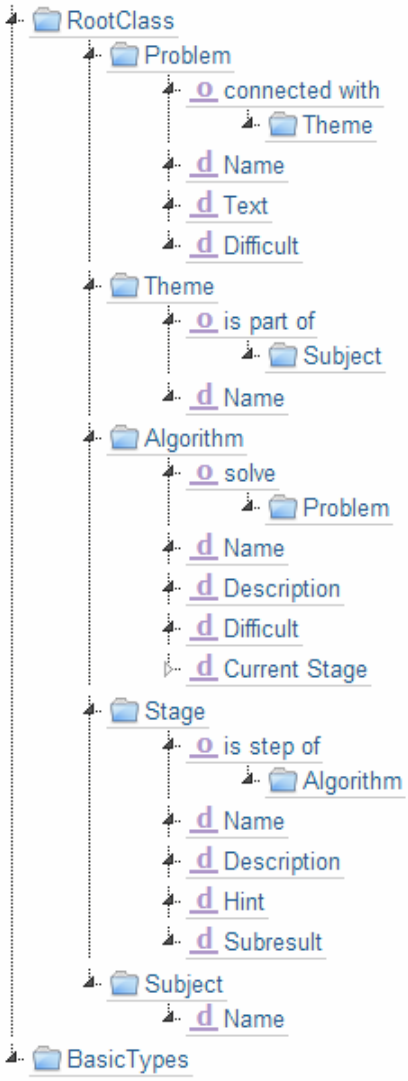

Рис. 3 – Созданная онтология

В разработанной онтологии присутствуют объекты Алгоритм (Algorithm) и Стадия (Stage), между ними следующая связь: Стадия – шаг Алгоритма (Stage is step of Algorithm). Каждый класс онтологий определяется связями с другими классами онтологии. Так, к примеру, к одной теме может относится множество задач.

Создан web-сайт, использующий спроектированную онтологию для хранения информационных ресурсов.

В настоящий момент кроме организации информационных ресурсов (задач, алгоритмов), их хранения и редактирования, реализована классификация информационных ресурсов, их сортировка по уровню сложности, а также поисковый сервис с возможностью фильтрации задач по теме и дисциплине.

Разработка системы будет продолжена, с целью создания семантических сервисов, использующих созданную базу знаний для решения задач адаптивного управления веб-информационной системой.

Так же будут созданы сервисы, позволяющие получать подсказки в ходе решения какой-либо задачи.

Литература

1. Грегер .С. Э. Администрирование и интерфейс пользователя CMSPlone / С. Э. Грегер // Федер. агентство по образованию, ГОУ ВПО «УГТУ-УПИ им. первого Президента России Б.Н. Ельцина», Нижнетагильский технол. ин-т (фил.). – Нижний Тагил: НТИ (ф) УГТУ-УПИ, 2009.-140с.

2. Грегер С. Э. Сервер приложений "Zope". Учебное пособие для вузов. – М.: Горячия линия – Телеком, 2009. – 256 с.: ил.

УДК 681.3

УПРАВЛЕНИЕ СТРУКТУРОЙ И ВИЗУАЛЬНЫМ ОТОБРАЖЕНИЕМ СИСТЕМЫ УПРАВЛЕНИЯ УЧЕБНЫМ ПРОЦЕССОМ НА ОСНОВЕ СЕМАНТИЧЕСКОЙ СЕТИ

Шадрина Екатерина Валерьевна, студент, ГОУ ВПО «Уральский федеральный университет имени первого Президента России Б.Н.Ельцина», Нижнетагильский технологический институт (фил.). Факультет экономики и менеджмента, кафедра информационных технологий, Россия, Нижний Тагил, Shadrina.ekaterina@inbox.ru

Назаров Максим Александрович, студент, ГОУ ВПО «Уральский федеральный университет имени первого Президента России Б.Н.Ельцина», Нижнетагильский технологический институт (фил.). Факультет экономики и менеджмента, кафедра информационных технологий, Россия, Нижний Тагил, Sin90gradus@gmail.com

Грегер Сергей Эдуардович, доцент, ГОУ ВПО «Уральский федеральный университет имени первого Президента России Б.Н.Ельцина», Нижнетагильский технологический институт (фил.). Факультет экономики и менеджмента, кафедра информационных технологий, Россия, Нижний Тагил, segreger@gmail.com

Учебный процесс – система организации учебно-воспитательной деятельности, в основе которой – органическое единство и взаимосвязь преподавания и учения; направлен на достижение целей обучения и воспитания. Определяется учебными планами, учебными программами, а также планами воспитательной работы соответствующих учебных заведений, включает все виды обязательных учебных занятий (уроки, лекции, семинары, лабораторные занятия, учебную и производственную практику) и самостоятельной работы учащихся. Повышение эффективности управления учебным процессом становится одним из направлений совершенствования деятельности учебного учреждения в целом. Под интеллектуальной системой обычно понимается любое программное средство, в архитектуре которого явно выделена база знаний [1]. Информационные системы, основанные на знаниях, строятся как специализированные оболочки над системой знаний, сформулированных в терминах некоторой онтологии. Онтологию O можно определить как кортеж:

O=<C, I, R, F>,

где:

C — множество концептов онтологии;

I— множество индивидуалов концептов онтологии;

R— множество бинарых связей между индивидуалами и концептами онтологии;

F — множество функций интерпретации.

Создавая семантический портал для системы электронного управления учебным процессом (СУУП), сформулируем задачи данной системы:

1. создание модели учебного процесса и представление этой модели в виде онтологии;
2. онтология является расширяемой в рамках каждой предметной области;
3. поддержкой онтологии занимаются преподаватели-разработчики курсов;
4. знания представлены в виде веб-страниц;

Создание единой модели для детального описания учебного процесса является весьма трудоемкой задачей. Решением этой проблемы может быть отказ от глубокой декомпозиции системы и включение в онтологию только наиболее значимых понятий из рассматриваемых предметных областей и создание для каждого из них детальных специализированных онтологий. В качестве онтологии будем использовать онтологию предметной области «Учебная деятельность» совместно с онтологией задач этой предметной области. Для эффективного управления онтологией и ее использования в веб-приложениях принято

решение о представлении онтологии в виде семантической сети, хранимой в объектной базе данных [2, 3]. Онтология предметной области должна отражать виды деятельности и формальные описания сущностей, включенных в процессы обучения, определять содержание образовательных процессов в основных классах. Для того чтобы осуществить удобную и понятную структуру хранения данных, к основным классам были добавлены подклассы. Например, для класса "Участник" были созданы формы, виды, технологии обучения; специальности; дисциплины; роли "студент" и "преподаватель" и т.д. В ходе работы, для создания индивидуальных учетных записей, были выделены подклассы "Сотрудники" и "Студенты". Для создания расписания и учебных планов преподавателей были созданы такие дополнительные подклассы, как: "День недели", "Вид недели", "Номер пары", "Занятие", "Вид занятия", "Номер аудитории", "Преподаватель" и пр. Для удобного обращения к объектной базе данных была разработана семантическая сеть классов – онтология, на основе структуры описанной системы, которая представляет собой информационную модель предметной области, имеющую вид ориентированного графа (Рисунок 1). В семантической сети роль вершин выполняют понятия базы знаний, а дуги (причем направленные) задают отношения между ними.

Онтология состоит из двух групп классов – классов предметной области и классов классификаторов. Для пар классов из этих двух групп устанавливается связь: классификатор относится к определенному классу.

Набор объектов классификатора задает разбиение множества объектов класса предметной области на группы. Набор классификаторов вместе с их взаимными связями задают класс шаблона отношений, поскольку структура отношений их объектов задает структуру связей между объектами классов предметной области. Между классификаторами устанавливается система наследования, а между шаблонами – система отношений специализаций шаблонов.

Интерфейс пользователя – центральный элемент любой современной программной системы. Снижение стоимости его разработки достигается использованием декларативного языка спецификации и автоматической генерации интерфейса. Именно пользовательский интерфейс является той составляющей программной системы, которая подвержена частым изменениям из-за наличия широкого круга пользователей с различным уровнем подготовки и требованиями к программной системе. Реализация пользовательского интерфейса, удовлетворяющего требованиям пользователей, возможна только путем создания макетов будущего интерфейса и чем больше таких макетов в единицу времени может быть построено, тем больше вероятность построения "дружественного пользовательского интерфейса". Автоматизация создание таких макетов может быть достигнута созданием их высокоуровневых декларативных описаний с последующей автоматической генерацией (полной или частичной) программного средства, производимой специальным программным агентом визуализации. Для эффективного сопровождения изменений, вносимых в семантическую модель агент визуализации должен позволять каждому термину модели предметной области автоматически сопоставлять интерфейсные элементы, нужные для представления модели и заполнения. При этом в случае изменения модели онтологии не будет возникать необходимости вносить изменения в пользовательский интерфейс, так как они производятся автоматически.

Для разработки данной системы был выбран OntoEditor — продукт для Plone[4], созданный для работы с онтологиями, их классами, объектами классов, свойствами и т.п. Продукт позволяет пользователю в визуальном режиме создавать и расширять онтологию учебного процесса и хранить ее в виде семантической сети объектов объектной базы данных. Его описание можно найти на сайте https://www.academia.edu/3993991, http://yadi.sk/d/Ug3J3OG8CwVhS (статья: проектирование и реализация онтологии навигационной системы сайта (стр. 19)).

Для решения задачи автоматической генерации форм интерфейса пользователя разработан агент визуализации. Агент (концептуальная модель, модель задач, модель

коммуникаций с компонентами системы управления), представленный в виде семантической сети (Рисунок 2). На рисунке представлена связь между классами различных онтологий и связи между объектами различных онтологий, представляющие собой иерархическую структуру на плоскости.

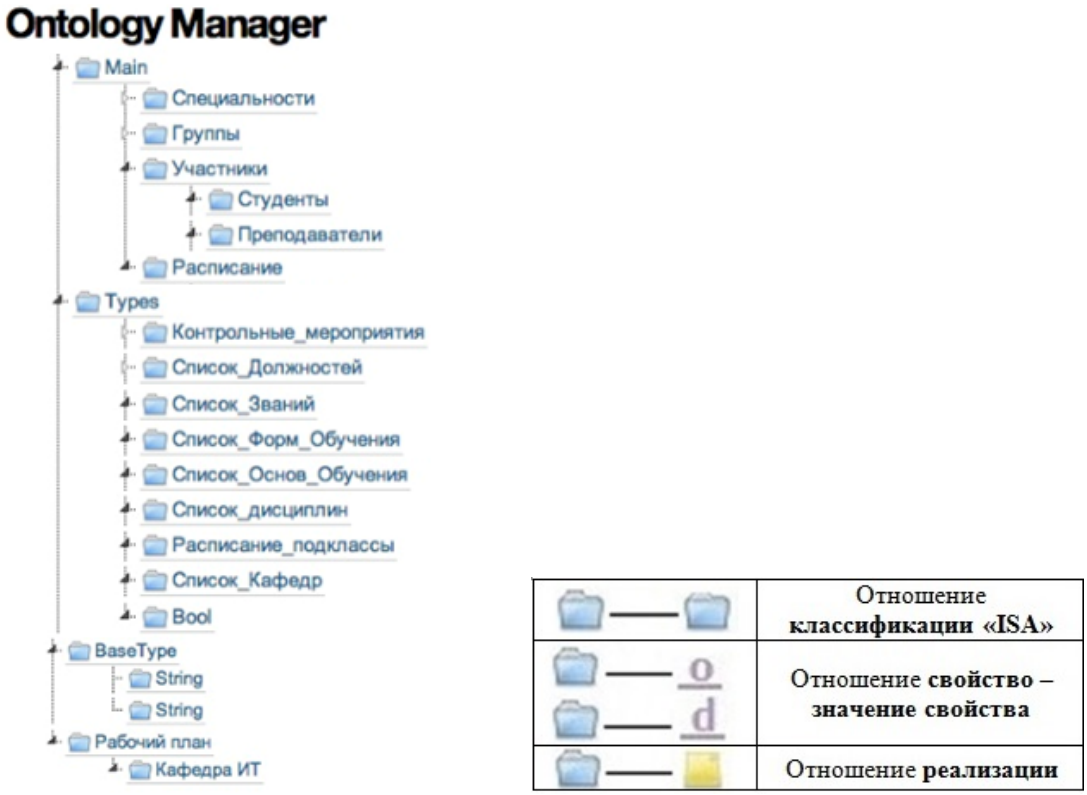

Рис. 1 – Структура онтологии

Представление агента в виде семантической модели и включение модели в онтологию интеллектуальной системы позволяет в дальнейшем расширять возможности системы по генерации интерфейса пользователя путем изменения модели агента.

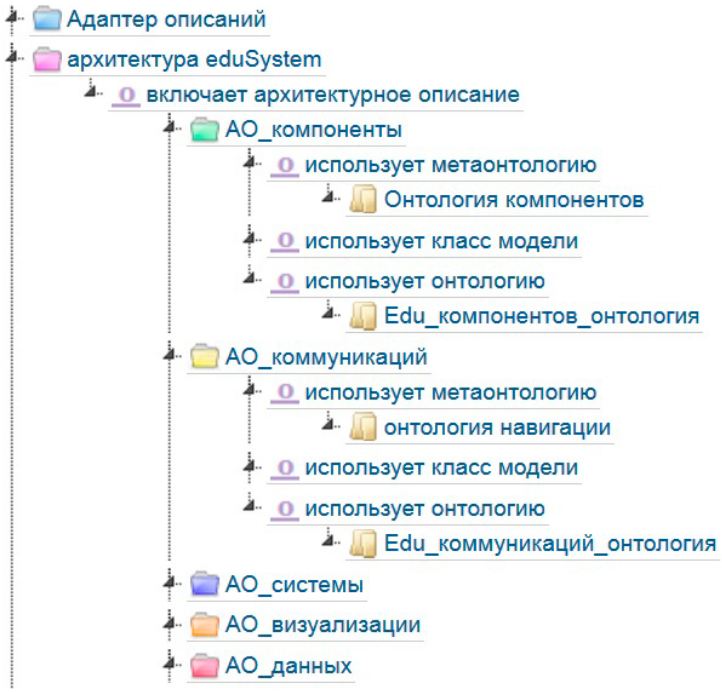

Рис. 2 – Концептуальная модель построителя форм

Процесс создания шаблонов вывода так же можно условно разделить на несколько этапов: получение списка всех объектов одного класса; получение всех свойств и связей для каждого из объектов; получение значений свойств и связей соответствующих объектов; структурирование информации в удобный для понимания вид. Пример автоматически созданного шаблона для создания объекта класса «Сотрудник» представлен на рисунке (Рисунок 3).

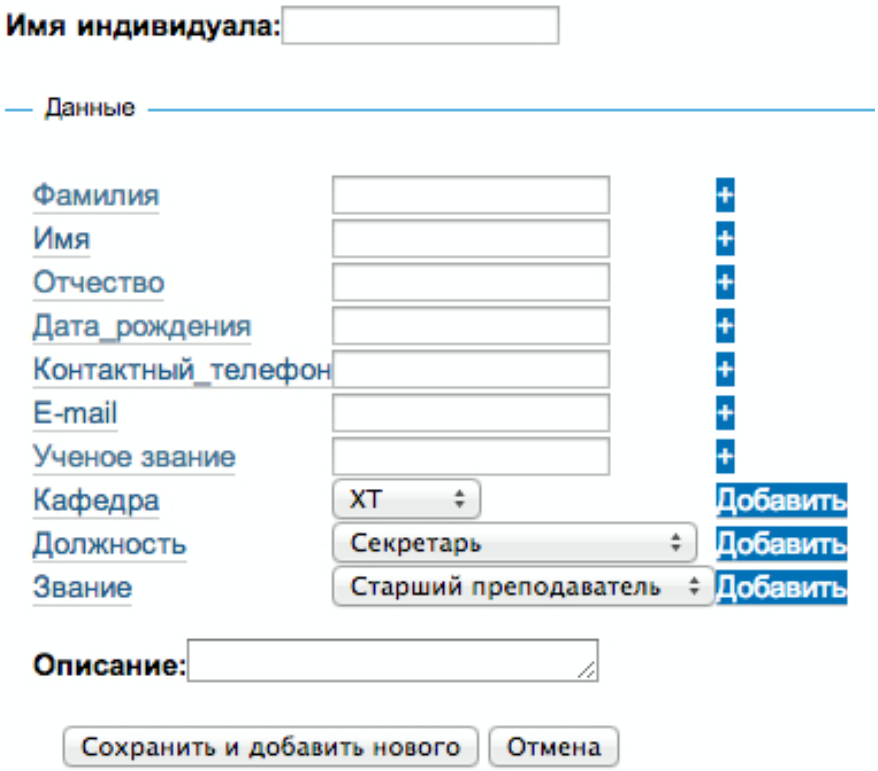

Рис. 3 – Шаблон добавления элемента на примере добавления элемента в класс "Сотрудник"

В результате проделанной работы была спроектирована интеллектуальная информационная система управления учебным процессом. Спроектированная система имеет простой и удобный интерфейс, позволяет менять модель предметной области и хранить данные и автоматически генерировать пользовательский интерфейс. Достоинством данной системы является легкая масштабируемость в случае увеличения подразделений учебного учреждения, а также возможность удаленного использования посредством сети Интернет.

Литература

1. Клещев А.С., Шалфеева Е.А. Грибова В.В. Системы управления интеллектуальными Интернет-приложениями // Владивосток:ИАПУ ДВО РАН. — 2010. — с. 31.
2. Поршнев С.В. Грегер С.Э Совместное использование онтологической модели и объектной моделей при проектировании и реализации информационных WEB-систем // Естественные и технические науки. — №6. — 2011. — с. 461-468
3. Сковородин Е.Ю. Грегер С.Э. Построение онтологического портала с использованием объектной базы // Объектные системы – 2010: Материалы I Международной научно-практической конференции. — Ростов-на-Дону, 2010 г. — с. 74-78.
4. Грегер С.Э. Администрирование и интерфейс пользователя CMS Plone (монография) — Нижний Тагил: Федер. Агентство по образованию, ГОУ ВПО "УГТУ-УПИ им.первого Президента России Б.Н.Ельцина". Нижнетагил. технол. ин-т (фил.) НТИ(ф) УГТУ-УПИ. – 140с, 2009.

УДК 81.139, 004.041, 004.272.44

ОБЪЕКТНО-АТРИБУТНЫЙ ПОДХОД ДЛЯ СЕМАНТИЧЕСКОГО АНАЛИЗА ЕСТЕСТВЕННОГО ЯЗЫКА[1,2]

Салибекян Сергей Михайлович, доцент, Московский институт электроники и математики Национального исследовательского университета «Высшая школа экономики», Москва, ssalibekyan@hse.ru
Халькина Светлана Борисовна, студент, Московский институт электроники и математики Национального исследовательского университета «Высшая школа экономики», Москва, sv.halkina@gmail.com
Тиновицкий Кирилл Дмитриевич, студент, Московский институт электроники и математики Национального исследовательского университета «Высшая школа экономики», Москва, craabpinguin@yandex.ru

В статье рассматривается применение объектно-атрибутного (ОА) подхода для семантического (смыслового) анализа естественного языка (ЕЯ). Проблематика семантического анализа становится актуальной все больше и больше: объем информации в мире растет экспоненциальными темпами и применяемый в настоящее время поиск по ключевым фразам уже не может дать пертитентных (соответствующих замыслу пользователя) результатов; мировая глобализация требует все больших объемов автоматического перевода с иностранных языков, а применяемые сейчас статические алгоритмы не обеспечивают должного качества перевода; все более актуальным становится автоматическое реферирование текстов и т.д. За 50-летнюю историю компьютерной лингвистики было предложено множество методик семантического анализа текста: генеративные грамматики, теория «Смысл <–> текст» Игоря Мельчука, концепция семантического Web. Однако до сих пор не разработано методики, описывающей семантический анализ ЕЯ на всех стадиях анализа, обеспечивающей единый подход буквально на всех все уровнях организации вычислительной системы и всех стадиях анализа языка. Предлагаемый нами подход, помимо других достоинств, лишен этого недостатка.

ОА-подход [1] объединяет в себе буквально все стадии анализа ЕЯ и уровни вычислительной системы – формат данных, методика построения семантической сети и поиск информации в ней, правила преобразования исходного текста в семантическую сеть и язык программирования для описания алгоритма преобразования и т.д. Благодаря тому, что все уровни вычислительной системы работают по единым принципам, значительно повышается эффективность системы в целом. ОА-подход относится к классу dataflow (вычислительный процесс с управлением потоком данных) и обладает массой положительных качеств: объектный принцип, способность синтеза и модификации сложно структурированных данных непосредственно во время вычислительного процесса, удобство организации параллельных и распределенных вычислений и т.д., – которые позволяют эффективно реализовывать системы семантического анализа ЕЯ.

Как известно, применение объектно-ориентированной парадигмы (ООП) для распознания ЕЯ не дало ожидаемого эффекта. Так, ОО-парадигма не нашла широкого применения даже в области баз данных, чего же говорить о создании баз знаний (серьезные надежды на то, что ООП потеснит реляционную парадигму баз данных, улетучились еще в середине 1990-х годов) [2]. Причиной тому послужили громоздкость и негибкость ООП [3]: класс, как структура данных, создается заранее и не изменяется во время вычислительного процесса, таким образом, вычислительная система способна распознавать только те объекты, которые были заранее заложены в нее ее создателем; дополнение же базы знаний новой

[1] Статья рекомендована к опубликованию в журнале "**Информационные технологии и вычислительные системы**"
[2] Исследование осуществлено в рамках Программы фундаментальных исследований НИУ ВШЭ в 2014 году

информацией влечет за собой трудоемкий процесс рефакторинга (изменения внутренней структуры) программы. Получается, что с помощью ООП весьма затруднительно создавать системы искусственного интеллекта и системы семантического анализа ЕЯ в том числе, в задачу которых входит осмысливание новых, еще неизвестных компьютеру объектов, смысловых связей и фактов. Поэтому для реализации семантического анализа ЕЯ необходим другой подход. И наше исследование как раз и посвящено применению альтернативного ООП объектно-атрибутного (ОА) подхода к организации вычислительного процесса. Подход этот реализует объектный принцип [4], т.к. с его помощью можно описывать объекты, их поведение и их взаимодействие, обеспечивая таким образом абстракцию данных и программы. Однако, в отличие от ООП, ОА-подход дает возможность легкого синтеза и модификации информационных конструкций непосредственно во время вычислительного процесса по заложенным в вычислительную систему правилам (алгоритму). Еще одним преимуществом ОА-подхода можно считать разработанную для него теоретическую базу, описанную в [5].

Итак, опишем разработанную нами методику семантического анализа ЕЯ. Основу системы анализа составляет семантико-морфологический словарь, в который помещаются описания лексем. Описание одной лексемы представляет собой ОА-список [6] всех возможных ее толкований – как известно, лексемы ЕЯ обладают свойством полисемии (многозначности). Каждое толкование лексемы представляет собой совокупность по крайней мере двух связанных между собой информационных капсул (ИК): капсула с описанием морфологических свойств толкования лексемы (падеж, род, число и т.п.) и капсула с семантическими свойствами. При анализе текста происходит поиск лексем в этом словаре, и из найденных описаний формируется ОА-список толкований лексем исходного текста.

Далее происходит преобразование этого списка в семантическую сеть (или в нашей терминологии семантический ОА-граф), которая, по сути, будет представлять онтологическую базу знаний, сформированную исходя из информации, заложенной в анализируемом тексте. Такую семантическую сеть можно будет использовать для поиска необходимой информации, автоматического перевода, автоматического реферирования и т.п.

Преобразование осуществляется в несколько этапов (проходов) от простого к сложному. На каждом проходе происходит «склейка» второстепенных лексем в словосочетании (синтагме) с лексемой главной. Так, при анализе русского языка на первом проходе осуществляется склейка наречий степени с качественными наречиями и глаголами. Например, для словосочетания «очень зеленый» в капсулу с семантическим описанием слова «зеленый» добавится информационная пара (ИП) с атрибутом «степень» и нагрузкой, в которой будет записан код, определяющий степень «зелености» (для определения степени, например, можно воспользоваться лингвистической переменной [7]). Описание же слова «очень» удаляется из списка лексем исходного текста. Вторым проходом при анализе русского языка является склейка прилагательных с существительными. Например, в синтагме «очень зеленый стул» на первом проходе происходит склейка слов «очень» и «зеленый» (слово «очень» удаляется из списка и «повисает» на слове «зеленый») – в результате в списке исходных лексем остается словосочетание «зеленый стул», которое на втором проходе преобразуется так: к описанию слова «стул» добавляется атрибут «цвет» со значением «зеленый» и степенью «очень», а описание слова «зеленый» удаляется из исходного списка лексем. В результате, получается следующий семантический ОА-граф: {Слово=«стул» атрибут={**Цвет**=Зеленый **Степень**=Очень}}. Следует отметить, что при склейке к главному слову «прилипает» только ИК с семантическим описанием второстепенной лексемы, ИК с описанием морфологических свойств удаляется за ненадобностью, т.к. в дальнейшем не понадобится для семантического анализа. В итоге в ОА-граф, описывающий смысл текста, попадают только ИК с описанием семантики.

Далее следуют проходы для анализа синтаксических конструкций с союзами, и заключительный проход – склейка существительных и глаголов. Для анализа смысловых связей между предложениями в ОА-системе применяется так называемый тематический

словарь, представляющий собой ОА-список, куда помещаются описания тех объектов, упоминания о которых встречались в тексте ранее. Например, при анализе первого предложения из фрагмента текста «В комнате стоял **стул**. **Стул** был зеленым.» описание объекта «стул» попадет в тематический словарь; при анализе же второго предложения для слова «стул» будет найден объект, который уже упоминался в тексте и к нему будет добавлено свойство «зеленый». Схема синтеза семантического графа из текста на ЕЯ приведена на рис. 1.

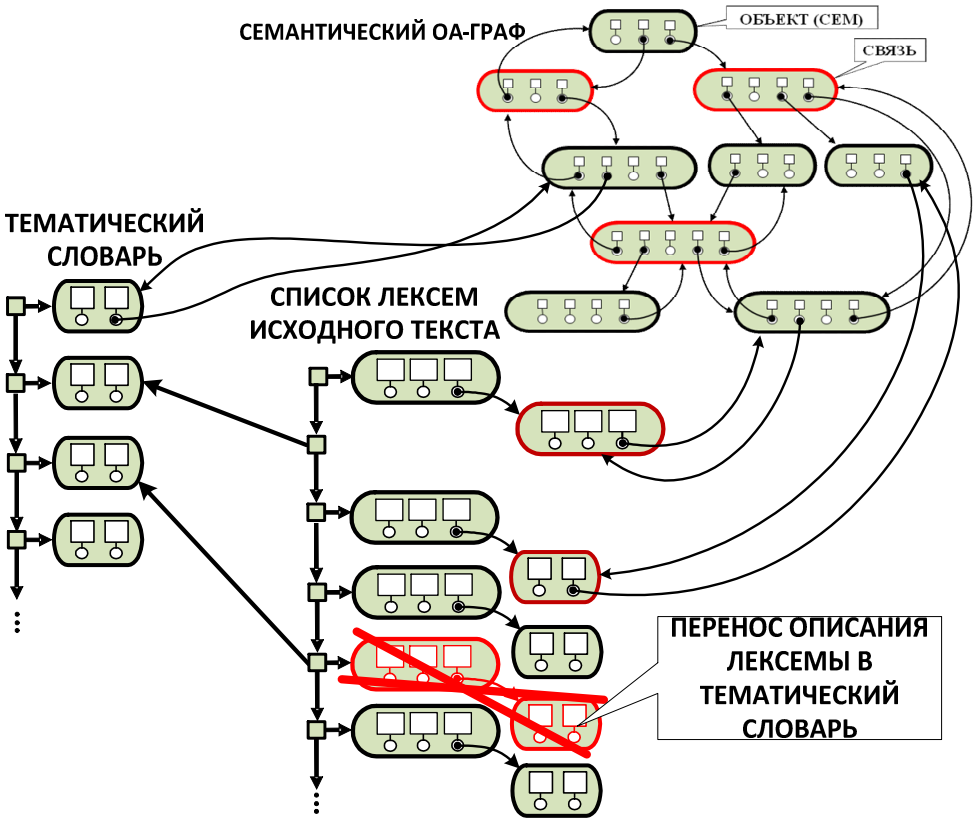

Рис. 1 – Схема построения ОА-графа из списка лексем исходного текста

Алгоритм преобразования текста в семантический граф задается с помощью правил преобразования (ОА-грамматика), основой которых стали формальные грамматики Хомского. ОА-грамматика в отличие от своего прототипа оперирует не с цепочкой символов, а с цепочкой (списком) капсул с описанием лексем (список исходных лексем) и служит для формирования семантического ОА-графа. Формально ОА-грамматику можно описать как четверку OAG = {A,L,P,G}, где A – алфавит атрибутов; L – алфавит нагрузок ИП (в этот алфавит входят не только числа и строки, но и ссылки на ИК; G – ОА-граф (список описаний лексем исходного языка); P – правила преобразования ОА-графа. Правило преобразования списка лексем состоят из двух частей (левая и правая), разделенных знаком «→». Слева помещается последовательность лексем из списка исходных лексем, справа – ОА-граф, на который заменяется последовательность этих лексем. ОА-граф описывается с помощью следующих обозначений:

– a = l – обозначение ИП (IP) ($a \in A$, $l \in L$, где A – множество атрибутов, L – множество нагрузок ИП);

– Name{…} – ИК (IC); между знаками «{» и «}» записываются ИП, входящие в ИК); ИК представляет собой цепочку ИП; Name – имя ИК (или ссылка на ИК);

– {…a_1= ICName{Ip_1,Ip_2…} …} – ИК, в которую входит ИП с указателем на другую ИК в нагрузке; имя ИК может и не указываться;

– * – конкатенация ИК (запись IC_1*IC_2 обозначение показывает, что капсула IC_2 должна быть сцеплена с капсулой IC_1);

– | – вариант (перечисление смысловых альтернатив);

– ( ) – группировка правил вариантов (знак «|» имеет более низкий приоритет, чем «*»), т.е. правила, заключенные в круглые скобки, имеют бóльший приоритет, например, выражение IC (IC2 | IC3) эквивалентно IC*IC2 | IC*IC3;

– :: – знак, показывающий, что ИК исключается из списка лексем.

Например, для обработки предлога «in» в английском языке применяются правила (цифрами обозначены 1-й и 2-й проходы анализа):

1. in NOUN → temp{Location ={Subj={NOUN*{Location=temp}} Location=in };

2. NOUN1 in*{ Location=temp} → NOUN1*{ Location=temp*{Obj=NOUN1}};

где Location – атрибут места расположения чего-либо, Obj – объект, Subj – субъект.

В результате применения вышеприведенных правил порождается фрагмент ОА-графа, состоящий из описаний двух объектов, связанных между собой пространственной связью «Находится в». На основе подобных правил строится ОА-программа, осуществляющая преобразования списка исходных лексем в семантический ОА-граф.

Теоретической основой семантического ОА-графа является теория семантических падежей (валентностей) Ч. Филмора [8], в которой вводится перечень возможных типов связей между объектами, описываемыми в тексте. Так, Филлмор выделяет 9 ролей (связей), а Ю.Д.Апресян – 25. Например: объект, субъект, инструмент. Так, для предложения «Мальчик ударил большую змею палкой» будет синтезирован следующий ОА-граф (жирным выделены атрибуты ИП, которые одновременно обозначают семантические роли/валентности):

{ **Объект** = мальчик
  Субъект = { **Объект** = змея **Свойство** = { **РамерОтносительный** = большой }
  }
  Действие = ударить
  Инструмент = палка
}

В настоящее время нами ведется работы по составлению перечня семантических ролей, т.к. ролей, выделенных Филлмором, Апресяном и другими лингвистами, не хватает для описания семантики текста. И уже выделено порядка сотни семантических ролей (валентностей).

Отдельно необходимо сказать о способе работы с полисемией (многозначностью) лексем, которая обеспечивается с помощью многоуровневого ОА-списка. На рис. 2 проиллюстрирована обработка двух лексем, имеющих по два толкования. Допустим, согласно правилам преобразования списка лексем, необходимо произвести склейку 2-го и 3-го толкований лексем. Тогда происходит так называемое расфоркивание списка толкований лексем (от англ. Fork – развилка), в результате которого образуется список из четырех альтернативных толкований фрагмента текста, содержащих все возможные комбинации толкований этих двух лексем. На ветви, соответствующей склеиваемым толкованиям, осуществляется модификация списка лексем, согласно применяемому правилу преобразования. Альтернативные ветви толкований фрагментов текста могут не только генерироваться в результате расфоркивания, но и удаляться из списка, если в процессе дальнейшего анализа обнаруживается синтаксическое или семантическое несогласование лексем в ней. Таким образом, преобразование списка толкований исходных лексем в семантический ОА-граф представляет собой процесс генерации и уничтожения альтернативных ветвей толкований фрагментов текста. Семантическое согласование заключается в сравнении семантических атрибутов. Например, после союза «в» («in») должно обязательно находиться слово с семантическим атрибутом «вместилище» («container»), после союза «на» («Under») обязательно должна находится лексема с семантическим атрибутом «поверхность» («surface») и т.д.

Получившийся семантический ОА-граф, состоящий из вершин трех типов – объекты (задаются существительными), связи между объектами (глаголы или союзы) и атрибуты/свойства (прилагательные), – может быть использован, например, для

семантического поиска. Для этого введенный пользователем запрос на ЕЯ также преобразуется в ОА-граф и далее происходит определение, является ли ОА-граф запроса подграфом ОА-графа текста; если да, то пользователю выдается сообщение об успешном поиске.

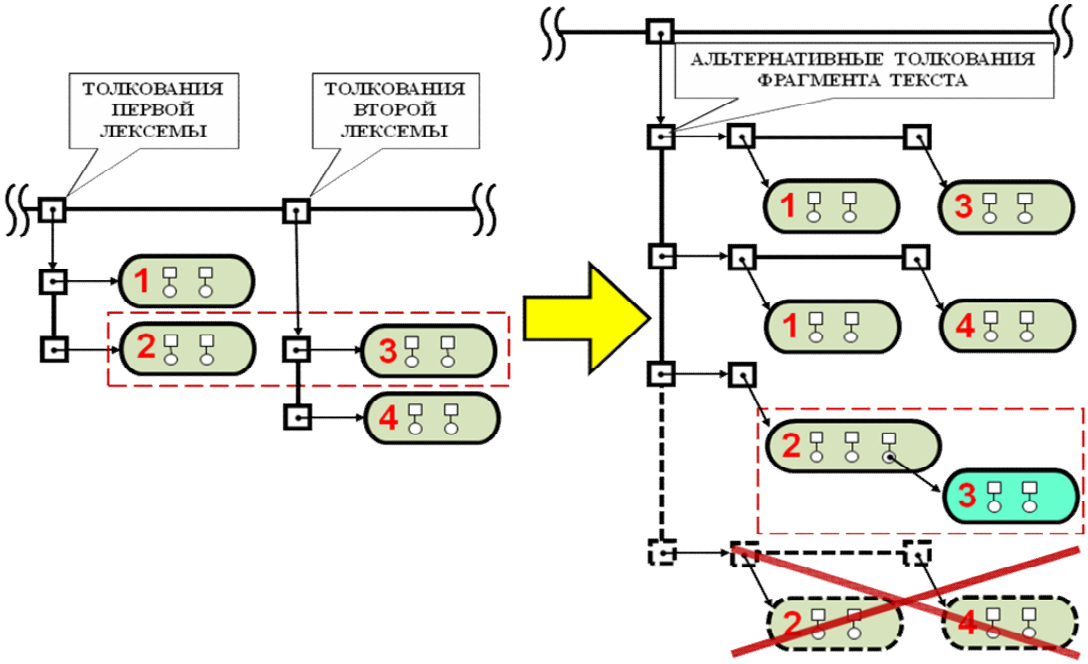

Рис. 2 – Работа с многозначностью лексем естественного языка

Для совпадения графов необходимо, чтобы ИП из капсул из вершин графа-запроса полностью совпали с ИП из капсул в вершинах графа-текста. Для ускорения поиска подграфа была разработана методика спектра атрибутов. По этой методике происходит подсчет всех атрибутов, встречающихся в ОА-графе, и далее поиск подграфа начинается с тех вершин, в которых обнаружено наименьшее количество совпадений в обоих ОА-графах. В ОА-графе используются двусторонние связи между узлами для того, чтобы можно было произвести обход графа начиная с любой его вершины. Таким образом, удастся значительно снизить число переборов во время поиска (рис. 3).

В настоящее время разработанные принципы анализа ЕЯ реализованы на практике – нами созданы экспериментальные базы знаний для анализа русского и английского языков, включающие в себя порядка двух сотен слов и в районе ста правил преобразования списка исходных лексем, что позволило доказать работоспособность предложенной методики семантического анализа ЕЯ.

В заключении следует сказать, что разработанная методика объединяет все уровни вычислительной системы. Дело в том, что вычислительная нагрузка по семантическому анализу ЕЯ ложится на функциональные устройства (ФУ). ФУ могут быть реализованы как программно, так и аппаратно, т.е. объектный принцип может быть реализован уже на уровне «железа», чего в настоящее время невозможно для ООП. Формат данных для всех этапов анализа ЕЯ совпадает – все данные и их структура представлены в виде ИП, ИК и ОА-графов. Благодаря этому отпадает необходимость в конвертации данных при переходе с одного уровня анализа ЕЯ на другой, и также сводится к минимуму семантический разрыв между языком программирования высокого уровня (для программирования ОА-системы применяется специализированный язык) и аппаратной частью компьютера. Работать в единой парадигме смогут буквально все специалисты, занятые в создании систем семантического анализа ЕЯ: аппаратчики, программисты, лингвисты, математики.

Также следует отметить, что формат представления программы в ОА-системе такой же, что и для представления данных (программа представляет собой последовательность ИП, которые передаются для ФУ) – и программу можно при необходимости встраивать в ОА-

граф, наделяя таким образом его поведением (в рамках проводимой работы для программного кода использовался отдельный ОА-граф).

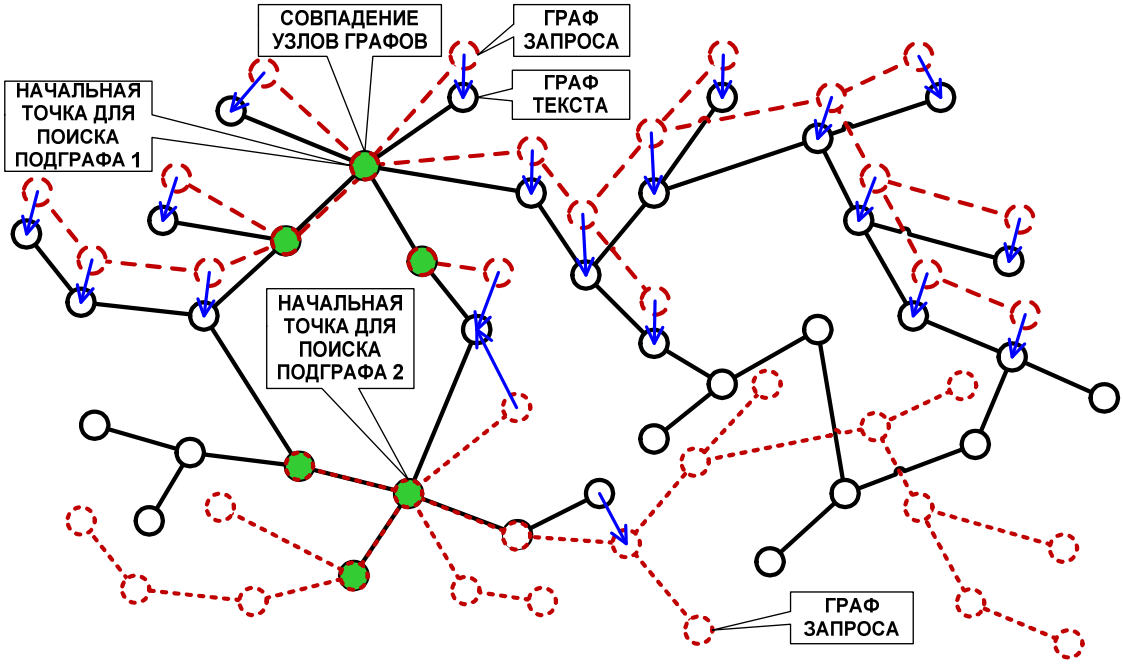

Рис. 3 – Поиск подграфа в семантическом ОА-графе текста

В дальнейших планах работ – расширение семантико-морфологического словаря и увеличение числа правил обработки списка лексем и критериев семантического согласования лексем в экспериментальной базе данных. В результате этой работы появится возможность обработки ОА-системой адаптированных текстов, что позволит провести имитационное моделирование ОА-системы с целью определения ее характеристик: сложность алгоритма обработки ЕЯ, требуемый объем памяти, быстродействие вычислительной системы.

Литература

1. Салибекян С.М., Панфилов П.Б. Объектно-атрибутный подход к построению интеллектуальных систем // Нейрокомпьютеры: разработки и применение. 2011, №11 – с. 9-17.
2. Дейт, К. Дж. Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильяме", 2005
3. Gabriel, R. Objects Have Failed: Notes for a Debate. (retrieved 17 May 2009). http://www.dreamsongs.com/Files/ObjectsHaveFailed.pdf
4. Салибекян С.М., Панфилов П.Б. ОА-архитектура – новый подход к созданию объектных систем // Объектные системы – 2011: материалы III Международной научно-практической конференции (Ростов-на-Дону 10-12 мая 2011 г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону, 2011. – С. 73-79 URL: http://objectsystems.ru/files/Object_Systems_2011_Proceedings.pdf
5. Салибекян С.М., Панфилов П.Б. Формализация dataflow-модели вычислительного процесса. // Объектные системы – 2013: материал V Международной научно-практической конференции (Ростов-на-Дону, 10-12 мая 2013 г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ ЮРГТУ (НПИ), 2013. – С. 87-93 URL: http://objectsystems.ru/files/2012/Object_Systems_2013_Proceedings.pdf
6. Салибекян С.М., Панфилов П.Б. Анализ языка с помощью вычислительной системы объектно-атрибутной архитектуры. // Объектные системы – 2012: материал VI Международной научно-практической конференции (Ростов-на-Дону, 10-12 мая 2012 г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ ЮРГТУ (НПИ), 2012. – С. 31-37 URL: http://objectsystems.ru/files/2012/Object_Systems_2012_Proceedings.pdf
7. Л. Заде. Понятие о лингвистической переменной и его применение к принятию решений. – М.: Мир, 1976.

8. Филлмор Ч. Дело о падеже // Новое в.зарубежной лингвистике. М.: Прогресс, 1981. Вып.X. С. 369-495.

УДК 04.004

ТЕСТОВАЯ МОДЕЛЬ ДЛЯ ОБУЧЕНИЯ ПРОЕКТИРОВАНИЮ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ БАЗ ДАННЫХ

Олейник Павел Петрович, к.т.н, системный архитектор программного обеспечения, ОАО "Астон", доцент, ШИ (ф) ЮРГПУ (НПИ) им. М.И. Платова, xsl@list.ru

Обучение студентов на высоком уровне и подбор простых и всеобъемлющих примеров – основная задача любого преподавателя ВУЗа. Ключевой проблемой в данном случае является нехватка времени, отводимого на практические занятия, что связано с наличием множества функциональных возможностей в современных языках программирования, средах разработки, систем управления базами данных. В статье представлена простая тестовая модель, используемая автором в течение нескольких лет при обучении студентов проектированию объектно-ориентированных баз данных (ООБД) при изучении соответствующей дисциплины. Представленная диаграмма классов может быть использована при обучении студентов на курсах, связанных с изучением объектно-ориентированной парадигмы.

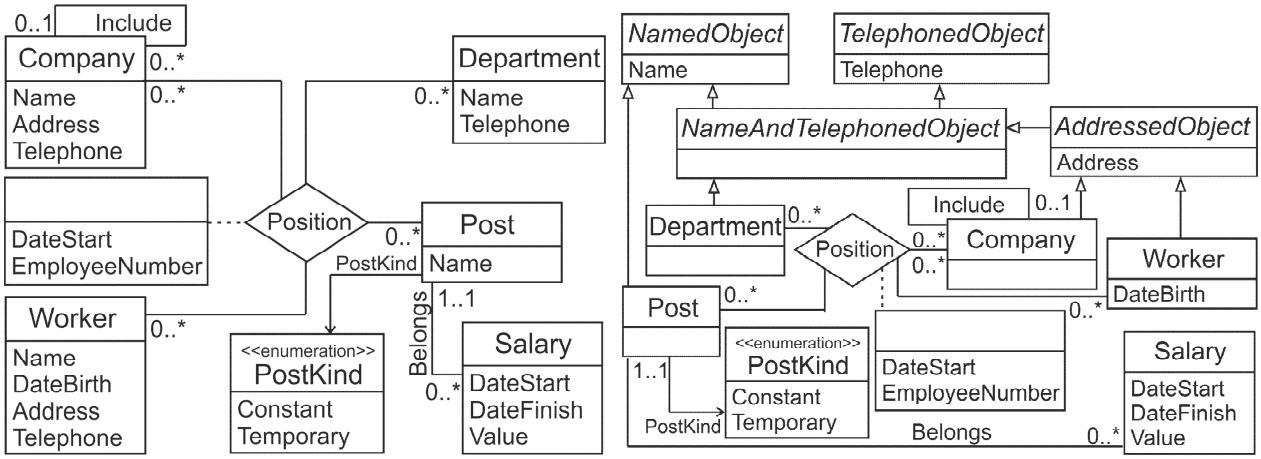

Рис. 1 – Концептуальная (слева) и логическая (диаграмма классов, справа) модели тестовой предметной области

В качестве предметной области использована информационная система для отдела кадров. Эта предметная область понятна каждому специалисту и довольно подробно описана в различных источниках, в том числе и работе, посвященной языку моделирования UML [1]. Т.к. реализовать структуру БД необходимо в среде ООСУБД, то финальной стадией проектирования является построение диаграммы классов, для которой были выделены следующие критерии оценивания (КО), требующие наличия:

1. Глубокой иерархии наследования;
2. Абстрактных и реализованных классов;
3. Сложных (n-арных) ассоциаций;
4. Связей с атрибутами;
5. Перечислений;
6. Рекурсивных связей.

Рассмотрим необходимость введения каждого критерия. Опыт преподавания показал, что выделить базовый класс и унаследовать от него производный может практически каждый студент. Однако построение правильной глубокой иерархии для многих является

сложной задачей. Поэтому в тестовой модели необходимо предусмотреть не менее трех классов, связанных отношением наследования.

При проектировании структуры БД студент детально изучает описание предметной области. Описание представляет набор предложений, представляющих бизнес-процессы. С точки зрения реализации описываются только экземпляры конкретных классов, но при проектировании необходимо выделить наиболее общее поведение, что приводит к появлению на диаграмме классов абстрактных классов, экземпляры которых не будут созданы в системе (в ООСУБД). Точное понимание различий между абстрактными и конкретными классами – это основа основ при проектировании, поэтому требование наличия таких сущностей оформлено в $КО_2$.

Функциональные возможности унифицированного языка моделирования UML позволяют описать не только простые бинарные ассоциации, соединяющие только два класса, но и более сложные (n-арные), соединяющие три и более класса. Именно для представления таких элементов введён $КО_3$. Особенно интересны подобные связи с точки зрения реализации в целевом языке программирования, так как приводят к появлению промежуточного класса.

Перейдем к $КО_4$. Выделение характеристик, относящихся к конкретным сущностям, в конечном итоге приводит к появлению атрибутов, принадлежащих к определенным классам. Опыт преподавания показал, что у студентов возникают некоторые проблемы при выделении атрибутов, относящихся к связям. Сложность состоит в том, что значения атрибутов относятся не к экземпляру сущности, а именно к экземпляру связи.

Очень часто необходимо предусмотреть набор заранее предопределенных констант, имеющих определенный смысл в контексте предметной области. Для подобных случаев вводятся перечисления, требование наличия которых представлено в $КО_5$.

Наличие рекурсивных связей предполагает присутствие отношений ассоциации, на обеих краях которой присутствует один и тот же класс. В $КО_6$ описано требование наличия подобных связей. Эти связи могут присутствовать во многих предметных областях и реализуются в виде древовидных структур.

Перейдем к процессу проектирования. Изучение процесса проектирования основывается на методологии «сущность-связь», описанной в работе [2]. Разработка структуры любой базы данных начинается с анализа, в ходе которого изучаются предметная область, основные бизнес-процессы, требования и пожелания будущих пользователей системы. Этот этап определяется с установления границ будущей разрабатываемой системы и определения функционала, который необходимо реализовать в будущем приложении.

Следующим этапом является концептуальное проектирование БД, под которым понимается конструирование информационной модели предприятия, независящей от каких-либо физических условий реализации. Концептуальное проектирование базы данных позволяет описать высокоуровневую модель и начинается с создания концептуальной модели данных предприятия, полностью независимой от любых деталей реализации. К последним относятся выбранный тип СУБД, состав программ приложения, используемый язык программирования, конкретная аппаратная платформа, вопросы производительности и любые другие физические особенности реализации.

Предпоследним этапом является логическое проектирование БД, под которым понимается конструирование информационной модели предприятия на основе существующих конкретных моделей данных, но без учёта специфики используемой СУБД и прочих условий реализации. В общем случае, логическое проектирование базы данных заключается в преобразовании концептуальной модели данных в логическую модель данных предприятия с учетом выбранного типа СУБД (в нашем случае – в понятия объектной СУБД). Логическая модель данных является источником информации для этапа физического проектирования. Она предоставляет разработчику физической модели данных средства проведения всестороннего анализа различных аспектов работы с данными, что имеет важное значение для выбора эффективного проектного решения.

Завершающим этапом является физическое проектирование базы данных, под которым понимается описание конкретной реализации базы данных, размещаемой во внешней памяти. Физический проект описывает базовые отношения, определяет организацию файлов и состав индексов, применяемых для обеспечения эффективного доступа к данным, а также регламентирует все соответствующие ограничения целостности и меры защиты.

Физическое проектирование базы данных предусматривает принятие разработчиком окончательного решения о способах реализации создаваемой базы. Поэтому физическое проектирование обязательно производится с учетом всех особенностей используемой СУБД. Между этапами физического и логического проектирования всегда имеется определенная обратная связь, поскольку решения, принятые на этапе физического проектирования с целью повышения производительности разрабатываемой системы, могут потребовать определенной корректировки логической модели данных.

Из представленного ранее видно, что процесс разработки структуры БД начинается с концептуального проектирования. На рис. 1 слева представлены все выделенные сущности, связи и атрибуты. Для описания организаций выделана сущность Company. С целью представления группы компаний и организации древовидной структуры используется рекурсивная связь Include. Сущность Department представляет отдел внутри организации. Для описания каждой должности выделена сущность Post. Тип должности (постоянная или временная) представлен перечислением PostKind. Информация о каждом работнике сохраняется в сущности Worker. Уровень заработной платы отображается в сущности Salary. Сложная связь Position представляет собой конкретную должность, занимаемую определенным сотрудником в указанном отделе указанной организации. При этом описанная ассоциация имеет атрибуты.

При построении логической объектно-ориентированной модели была детально проанализирована концептуальная модель. Оказалось, что многие сущности имеют одинаковые атрибуты сходного назначения. Например, атрибут Name, сохраняющий название, присутствует в сущностях Company, Department, Worker, Post, поэтому было принято решение выделить класс NamedObject. Так как в предметной области отсутствуют экземпляры данной сущности, то полученный класс является абстрактным.

При анализе концептуальной модели оказалось, что многие сущности имеют атрибут Telephone, позволяющий сохранить номера телефонов. В логической модели для описанных целей был выделен абстрактный класс TelephonedObject. Многие классы одновременно имеют атрибуты Name и Telephone. Чтобы избавиться от необходимости наследоваться от обоих классов объявлен промежуточный абстрактный класс NamedAndTelephonedObject.

Абстрактный класс AddressedObject содержит атрибут Address, который сохраняет адрес. Этот класс является базовым для классов Company и Worker. Остальные сущности переносятся с концептуальной модели практически без изменений.

Проверим полученную диаграмму классов на соответствие выделенным критериям оценивания. Из рис. 1 видно, что присутствует глубокая иерархия классов, т.к. класс Worker наследуется от класса AddressedObject, который унаследован от NamedAddTelephonedObject и выступает дочерним по отношению к абстрактным классам NamedObject и TelephonedObject. Следовательно, требования $КО_1$ выполнены.

Абстрактные классы, которые не отражают объекты предметной области, позволили упростить модель, и они имеются в достаточном количестве на построенной диаграмме классов. Это позволяет утверждать, что требования $КО_2$ выполнены.

Сложная ассоциация Position представляет собой n-арную связь и её наличие говорит о том, что требования $КО_3$ соблюдены. Требования $КО_4$ выполнены, т.к. имеется класс-ассоциация, имеющая атрибуты. В данном случае атрибуты DateStart и EmployeeNumber не относятся к какой-либо сущности, входящей в связь, а относятся именно к экземпляру ассоциации.

Условия $КО_5$ также выполнены, т.к. на диаграмме классов имеется перечисление. Рекурсивная связь представлена отношением ассоциацией Include, которая позволяет

описать группу компаний, что в итоге представится в виде иерархической структуры и соответствует требованию $КО_6$.

Таким образом, из описания видно, что представленная на рис. 1 (справа) диаграмма классов соответствует всем выделенным критериям оптимальности. Следующим шагом является реализация в среде целевого языка программирования и платформы. Студенты самостоятельно выбирают предпочитаемый язык программирования. Как правило, это либо C#, либо Java. При этом в качестве ООСУБД применяется бесплатная версия DB4O, в которой имеется набор библиотек для обоих перечисленных языков.

Литература

1. Новиков Ф.А., Иванов Д.Ю. Моделирование на UML. Теория, практика, видеокурс. – СПб.: Профессиональная литература, Наука и Техника, 2010. – 640 с.: ил. + цв. Вклейки (+ 2 DVD).
2. Коннолли, Томас, Бегг, Каролин. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание. : Пер. с англ. – М. : Издательский дом "Вильямс", 2003. – 1440 с. : ил. – Парал. тит. англ.

УДК 04.004

ОРГАНИЗАЦИЯ ПОРТАЛА НАУЧНОГО СОТРУДНИЧЕСТВА

Маликова Валерия Игоревна, ученица 8 «Б» класса, гимназии им. А. С. Пушкина города Шахты, Россия, Шахты, malikova@valeria.com
Олейник Павел Петрович, к.т.н, системный архитектор программного обеспечения, ОАО "Астон", доцент, ШИ (ф) ЮРГПУ (НПИ) им. М.И. Платова, Россия, Ростов-на-Дону, xsl@list.ru

Информационные технологии – это наука о хранении, обработке и передаче данных с применением персонального компьютера. Сейчас понятие «Информационные технологии» также известно как «Компьютерные технологии». Существующие естественные науки – химия, биология, физика и другие – состоят из отдельных дисциплин, но Информационные технологии объединяют множество отдельных дисциплин в одно целое для изучения общего предмета – информатики и вычислительной техники. К их числу относятся: теория информации, кибернетика, программирование, теория алгоритмов, искусственный интеллект и многое другое. Развитию Информационных технологий послужило одно из самых значительных достижений XX века – создание электронно–вычислительных машин – ЭВМ. Термины «компьютер» и «электронная–вычислительная машина» можно рассматривать как синоним.

Информационные технологии неразрывно связаны с понятием Интернет. Именно на примере глобальной сети Интернет можно проследить бурный рост компьютерных технологий, как в России, так и во всем мире. Создание портала научного сотрудничества является наиболее актуальной проблемой для нашей страны, т.к. они отсутствуют.

Интернет-портал – это объединение всех возложенных функций и инструментов сайтов в сети Интернет. С помощью интернет-портала все желающие смогут узнать о расположенных на территории региона научных и образовательных организациях, их исследовательских работах, научных сотрудниках, а также о достижениях нашего региона в мире наук. Интернет-портал не будет являться средством массовой информации. Портал будет способствовать развитию информатизации в сфере науки, созданию электронных образовательных сервисов.

Реализация подобного портала позволит как обмениваться актуальной научной информацией в области информационных технологий, так и осуществлять поиск коллег по интересам для организации совместного написания научных статей.

В России не существует полномасштабных научных порталов, а существует множество научных форумов, организованных на базе ВУЗов.

Задача создания эффективного научного Интернет–портала – разработать удобную систему публикации, подбора, отбора, анонсирования и рейтингования тематических статей, новостей, баз документов, фотографий и видео, создать все необходимые дополнительные инструменты общения, поиска, выбора, определения и направления интересов посетителя, группирование посетителей.

Иными словами, научный Интернет–портал – это Веб–сайт с огромным количеством представляемых услуг и сервисов, посещаемый большим количеством пользователей. Научных порталов мало – на каждом уровне (глобальном, национальном, региональном) их количество обычно измеряется единицами. Научный портал должен обладать: 1) достаточным объёмом информации, не перегружающей страницы; 2) удобным интерфейсом.

На данный момент в России как никогда остро стоит вопрос о необходимости создания научного портала. Это связано с тем, что научный портал даёт возможность быстрого и оперативного взаимодействия между учёными для плодотворной работы по различным областям научной деятельности. Разумеется, в настоящее время есть молодёжь, которая занимается научными исследованиями и для которых создаваемый научный портал просто необходим для того, чтобы они могли общаться с научными деятелями на уровне современных технологий.

Таким образом, разработка и внедрение научного портала повлечёт за собой создание современных информационных технологий в области развития научного потенциала нашего региона. Мы надеемся, что данная программа послужит стимулом для привлечения талантливой молодёжи к работе в научной сфере деятельности.

Для реализации полноценного портала необходимо выполнить следующие этапы:

1. Собрать контактные данные авторов статей.
2. Собрать информацию о научных интересах и написанных статьях.
3. Реализовать информационную систему и Веб-сайт, позволяющие формировать списки авторов и статей по определенной тематике.

С целью упрощения реализации портала необходимо начать с одной области научных знаний. Например, собрать информацию об авторах научных статей в области информационных технологий. На базе Шахтинского института с 2010 года регулярно проводится Международная научно-практическая конференция «Объектные системы», посвященная информационным технологиям, сайт которой находится по адресу http://objectsystems.ru/. По состоянию на 2013 год выяснилось, что:

1. Проведено 7 конференций;
2. Опубликовано более 150 докладов;
3. Имеется информация более чем о 500 авторов.

Этого вполне достаточно для начала работы над порталом. Остается реализовать только третий этап и разработать информационную систему и Веб-сайт, позволяющие формировать списки авторов и статей по определенной тематике.

Перейдем к практической реализации ядра портала научного сотрудничества. Диаграмма реализованных классов, построенная с применением языка UML, представлена на рисунке 1 [1].

Отметим, что представленная диаграмма классов является фрагментом, дополняющим соответствующие классы, выделенные для информационной системы каталогизирования научных работ при проведении международных конференций, описанной в работах [2-4].

Основная идея в том, чтобы создать структуру форумов, позволяющих создавать темы с привязкой к опубликованным статьям. Т.е. позволить напрямую задать вопрос автору статьи.

Корневым абстрактным классом является DesriptionedDatedFileObject, в котором имеется атрибут Description, позволяющий вводить описание тем. Атрибут DateTime сохраняет текущую дату и необходим для сортировки объектов при отображении. Атрибуты File1 и File2 позволяют сохранять вложения к сообщениям. Абстрактный класс Forum введен

для того, чтобы программно отличить созданный форум от сообщения внутри форума, представленное экземпляром класса ForumMessage. Реализованный класс NamedForum используется при создании тематических форумов, у которых пользователю можно задать любую тему, сохраняемую в атрибуте Name.

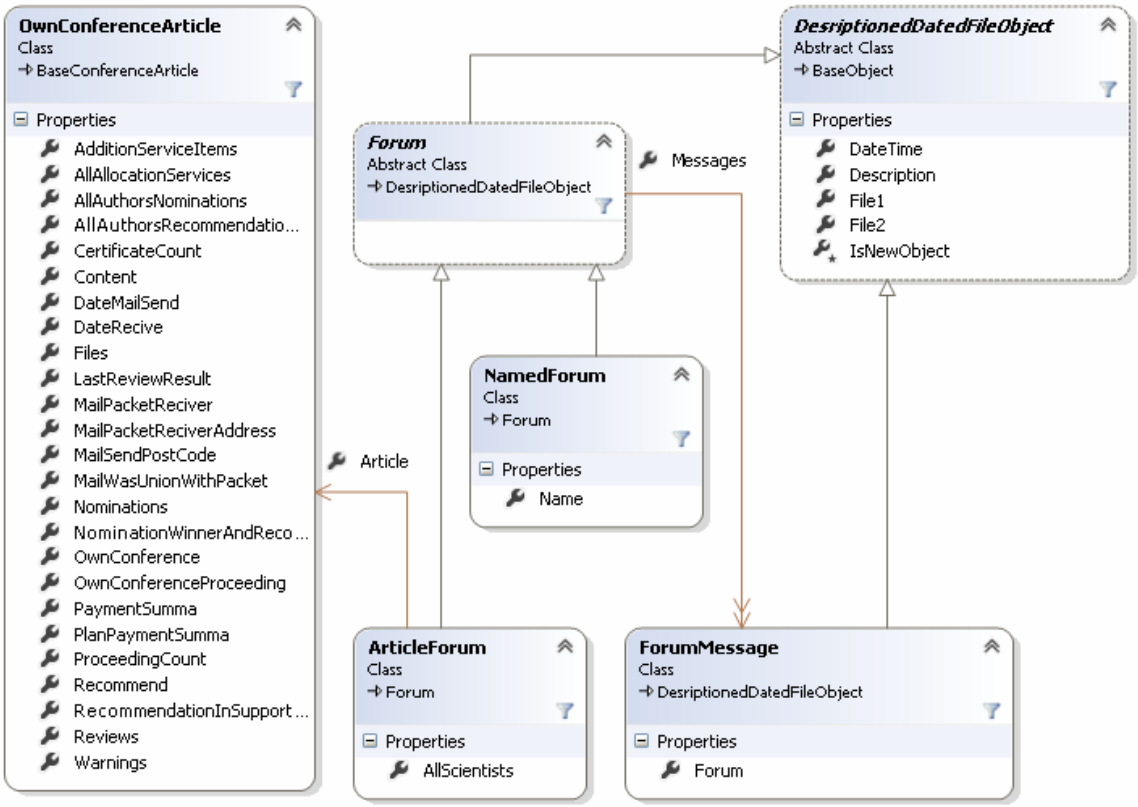

Рис. 1 – Диаграмма классов

Производный от Forum класс ArticleForum позволяет представить форум, посвященный обсуждению опубликованной статьи, ссылка на которую представлена атрибутом Article. При этом предполагается возможность обсуждения только статей конференции «Объектные системы», поэтому для описанного атрибута используется класс OwnConferenceArticle. Связь этого класса с классами, представляющими сборники конференции и со вспомогательными классами, описана в работах [2-4].

В данной статье описана необходимость создания единого портала научного сотрудничества и возможные пути решения. Одним из решения является создание Веб-форума на базе информации об авторах статей, опубликованных в рамках проведения международной научно-практической конференции «Объектные системы». В заключение изображена диаграмма классов, описаны состав и структура ее элементов.

Литература

1. Новиков Ф.А., Иванов Д.Ю. Моделирование на UML. Теория, практика, видеокурс. – СПб.: Профессиональная литература, Наука и Техника, 2010. – 640 с.: ил. + цв. Вклейки (+ 2 DVD).
2. Олейник П.П. Опыт применения инструментов объектно-реляционного отображения при разработке информационной системы каталогизирования научных работ // Информационные технологии в науке, экономике и образовании: материалы 5-й Всероссийской научно-практической конференции 2-3 сентября 2010 года / под. ред. О.Б. Кудряшовой; Алт. гос. техн. ун-т., БТИ. – Бийск: Изд-во Алт. гос. техн. уни-та, 2010. – С. 102-105.
3. Олейник П.П., Игумнов Е.А., Свечкарёв Е.А. Опыт проектирования информационной системы для каталогизирования научных работ при проведении международных конференций // Объектные системы – 2010: материалы II Международной научно-практической конференции. Россия, Ростов-на-Дону, 10-12 ноября 2010 г., Ростов-на-Дону, 2010. – С. 48-51.

4. Олейник П.П., Игумнов Е.А., Свечкарёв Е.А. Реализация модуля рецензирования в информационной системе проведения научных конференций // Объектные системы – 2011: материалы III Международной научно-практической конференции (Ростов-на-Дону, 10-12 мая 2011 г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону, 2011. – С. 26-29.

УДК 004.4'232+004.4'41

СЕМАНТИЧЕСКАЯ МОДЕЛЬ ПРОГРАММЫ И ЕЕ РЕАЛИЗАЦИЯ[1]

Лаптев Валерий Викторович, к.т.н., доцент, Астраханский государственный технический университет, Россия, Астрахань, laptev@ilabsltd.com
Грачев Дмитрий Александрович, аспирант, Астраханский государственный технический университет, Россия, Астрахань, dagrachev@list.ru

В работе [1] описан учебный язык программирования и реализация его интерпретатора в рамках интегрированной среды [2,3] для обучения программированию. Интегрированная среда строится вокруг редактора кода, который не является обычным текстовым редактором, а оперирует конструкциями языка программирования и объектами программы. Для реализации подобного подхода авторами разработана семантическая модель программы. В статье описана формальная семантическая модель, внутреннее представление модели в системе и ее реализация на языке C#.

Программа на учебном языке представляет собой либо единственный модуль, либо несколько модулей, один из которых является стартовым. Поэтому семантическая модель программы – это семантическая модель модуля. Модуль включает список импортируемых модулей, секцию определения и секцию инициализации. Секция инициализации содержит последовательность операторов, исполняемых при загрузке модуля в память. В секции определения задаются константы и переменные, определяемые типы, процедуры и функции. В общем случае все части модуля могут быть пустыми.

Для определения формальной семантической модели программы нужно определить формальную семантическую модель оператора. В общем виде формальная семантическая модель оператора представляет собой пару:

```
имя = < параметры, правила >
```

Имя – это уникальный идентификатор оператора. В учебном языке каждый оператор начинается ключевым словом, поэтому имя – это ключевое слово оператора. Параметры и правила определяют семантику оператора. Состав параметров зависит от оператора, набор правил представляет собой множество логических выражений и условий, которые должны быть истинными. Эти выражения и условия проверяются редактором при вставке оператора в программу.

В качестве примеров рассмотрим несколько формальных моделей простых и блочных операторов учебного языка [1]. Простыми операторами являются оператор импорта, операторы определения констант и переменных, операторы ввода и вывода, оператор присваивания и операторы вызова подпрограммы. К блочным операторам относятся оператор определения модуля, операторы определения подпрограмм, оператор цикла, условный оператор и оператор определения типа. Сначала опишем семантику оператора неформально, а затем приведем формальную модель.

Оператор определения переменной включает тип переменной и ее имя. Если тип переменной является простым (целый, вещественный, символ, булевский), то разрешается инициализировать переменную начальным значением. Инициализатор – это выражение, тип значения которого должен совпадать с типом переменной. Имя переменной не должно совпадать с ранее объявленными именами. Тип переменной должен быть либо

[1] Статья рекомендована к опубликованию в журнале "**Информационные технологии**"

определенным в языке, либо определенным в программе. Таким образом, формальную модель оператора объявления переменной можно представить следующим образом:

```
variable =
< param = < type, name, value >,
  rules = < ~(name ϵ Names), (type ϵ Types),
            if ~(value=null) then
              (type=integer)|(type=real)|(type=boolean)|(type=character)
          >
>
```

В этой модели определены следующие элементы:

- variable – ключевое слово оператора определения переменной в английской нотации;
- param – параметры формальной модели оператора;
- rules – правила и условия;

Параметрами являются:

- type – тип переменной;
- name – имя переменной;
- value – инициализирующее выражение переменной.

Модель включает два логических выражения:

- ~(name ϵ Names) – это выражение истинно при отсутствии имени определяемой переменной в множестве Names объявленных имен;
- (type ϵ Types) – это выражение истинно, если тип переменной присутствует в множестве Types встроенных или определенных типов программы;

В модели определено одно условие, при истинности которого проверяется истинность вложенных выражений и условий:

```
if ~(value=null) then
```

Это условие определяет, какие выражения должны быть истинны, если в определении переменной присутствует инициализирующее выражение. В данном случае выражение

```
(type=integer)|(type=real)|(type=boolean)|(type=character)
```

является истинным, если тип переменной является простым.

Блочные операторы отличаются тем, что включают тело – последовательность операторов учебного языка, которую обозначим как P:

$P = m_1\ m_2 \ldots, m_k,$

где k – произвольное целое положительное число. Последовательность операторов является обязательным параметром блочного оператора. Однако для разных блочных операторов на состав операторов в теле накладываются семантические ограничения. Например, оператор определения типа разрешается задавать только в секции определений модуля, как и операторы определения процедур и функций. Поэтому в модели блочного оператора требуется для параметра P указывать ограничения.

Определим множества S, элементами которых являются модели операторов, которые разрешается задавать в теле того или иного блочного оператора:

- S_{exe} = { constant, variable, let, input, output, call }
- S_{dcl} = { constant, variable, type, procedure, function }
- S_{imp} = { import }

Множество S_{exe} включает операторы, которые разрешается задавать в теле операторов определения процедур, функций, методов, в секции инициализации модуля, в теле оператора цикла и условного оператора. Операторы объявления константы и переменной включены в это множество, поскольку с помощью них во время выполнения определяются локальные в блоке переменные и константы. С помощью множество S_{dcl} определяет состав операторов объявления, которые разрешается задавать в секции определения модуля. И множество S_{imp}

определяет возможности конструирования последовательности операторов импорта. Тогда формальную модель оператора определения модуля можно задать таким образом:

```
module =
< param = < name, P_imp, P_dcl, P_exe >
  rules = < ~(name, ModuleNames),
              (P_imp ∈ S_imp)&(P_dcl ∈ S_dcl)&(P_exe ∈ S_exe)
          >
>
```

Эта модель устанавливает, что параметрами оператора определения модуля является имя модуля и три последовательности операторов: последовательность операторов импорта, последовательность операторов объявления в секции определения, последовательность исполняемых операторов в секции инициализации. Правила истинны, если операторы последовательностей принадлежат соответствующим множествам и имя модуля не должно совпадать с именами системных и входящих в программу других определенных модулей.

Из моделей блочных операторов самой простой является модель оператора цикла:

```
while =
< param = < exp, P >
  rules = < (type(exp)=boolean), (P ∈ S_exe) >
>
```

Параметрами, очевидно, являются булевское выражение – условие цикла, и тело оператора. Более сложную модель имеет условный оператор, поскольку для условного оператора требуется определить еще модели вложенных ветвей else_if и else:

```
else_if =
< param = < exp, P >
  rules = < (type(exp)=boolean), (P ∈ S_exe) >
>

else =
< param = < P >
  rules = <(P ∈ S_exe) >
>
```

Определим множество операторов, которые могут входить в тело условного:

$S_{if} = S_{exe} \cup \{ else\_if, else \}$

Тогда семантическая модель условного оператора определяется так:

```
if =
< param = < exp, P >
  rules = < (type(exp)=boolean), (P ∈ S_if), (last(P)=else) >
>
```

Третье семантическое правило устанавливает, что ветка else должна быть последней в теле.

Таким образом, семантическая модель программы определяется формальной моделью оператора определения модуля, который включает последовательность операторов импорта, операторов объявления и последовательность исполняемых операторов в секции инициализации. Тела блочных операторов, входящих в эти последовательности, содержат новые последовательности операторов.

Внутренним представлением семантической модели программы в системе является многоуровневый список списков. Каждый оператор программы представлен узлом списка, в котором сохраняется полная информация о семантике (в частности, каждый узел имеет внутреннее имя, соответствующее оператору). В каждом узле имеются ссылки на следующий и предыдущий. Комментарий в языке Semantic Language тоже является оператором, поэтому представляется отдельным узлом списка. В узлах, соответствующих блочным операторам, имеется ссылка на тело, которое представляет собой дочерний список данного узла. Соответственно, первый оператор дочернего списка имеет ссылку на родительский узел.

Эту структуру строит редактор кода в соответствии с действиями программиста. При загрузке готового проекта строится такая же структура. Мы называем такое представление *семантическим деревом* программы.

Первоначально (в момент включения в проект первого модуля, в котором еще нет ни одного оператора) дерево состоит из одного специального пустого узла, который имеет внутренне имя NullOperator. Этот же узел представляет в дереве пустые строки, которые вставляются в программу для повышения читабельности. Обычно новый оператор вставляется в программу на месте некоторой пустой строки. В дереве эта операция представляет собой простую замену узла NullOperator на конкретный узел-оператор. Если вставляемый оператор является блочным, то на месте тела также вставляется NullOperator. Например, после вставки первого оператора **модуль**, дерево выглядит так, как показано на рис. 1 (показаны связи только вперед).

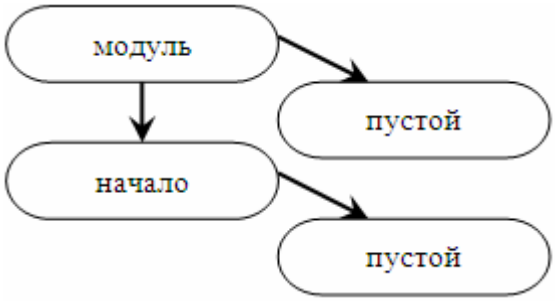

Рис. 1 – Семантическое дерево при вставке первого оператора **модуль**

Дочерняя вершина узла **модуль** – это тело секции определений, а дочерний узел вершины **начало** – это тело секции инициализации. Эти пустые узлы будут заменены на конкретные узлы-операторы при вставке. В качестве примера приведем простую программу, вычисляющую все факториалы от 1 до N.

```
#Вычисление факториалов#
модуль Факториал
    подключить Математика;

    закр процедура(входной целое x) Факт
        переменная-целое К;
        переменная-целое тФакт;
        присвоить тФакт := 1;
        присвоить К := 1;
        пока К <= x повторять
            вывести тФакт;
            вывести '\n';
            присвоить К := К + 1;
            присвоить тФакт := тФакт * К;
        конец цикла;
    конец Факт;

начало
    вызвать Факт(15);
конец Факториал.
#Конец модуля вычисления факториалов#
```

В этой простой программе имеются в наличии все основные конструкции, которые могут быть представлены в семантическом дереве: комментарии, пустые строки, простые и блочные операторы. Дерево, которое построил редактор по этой программе, показано на рис. 2 (показаны связи только вперед).

Решение о явном представлении программы в виде семантического дерева привело к важным последствиям при реализации интегрированной среды Semantic IDE [2,3]. Во-первых, регулярная структура семантического дерева делает алгоритм обхода [1] тривиальным. Во-вторых, ошибки выявляются в редакторе, поэтому на вход интерпретатора

поступает правильная программа. Интерпретатор при обходе дерева не выполняет ни лексического, ни синтаксического анализа – это существенным образом упростило реализацию, и повысило быстродействие. Вместо интерпретатора можно достаточно легко реализовать конвертер в семантически эквивалентный императивный язык программирования (например, в C или в Component Pascal) – для этого достаточно в алгоритме обхода вызов интерпретирующей функции заменить на вызов функции, формирующей текстовое представление оператора на языке программирования.

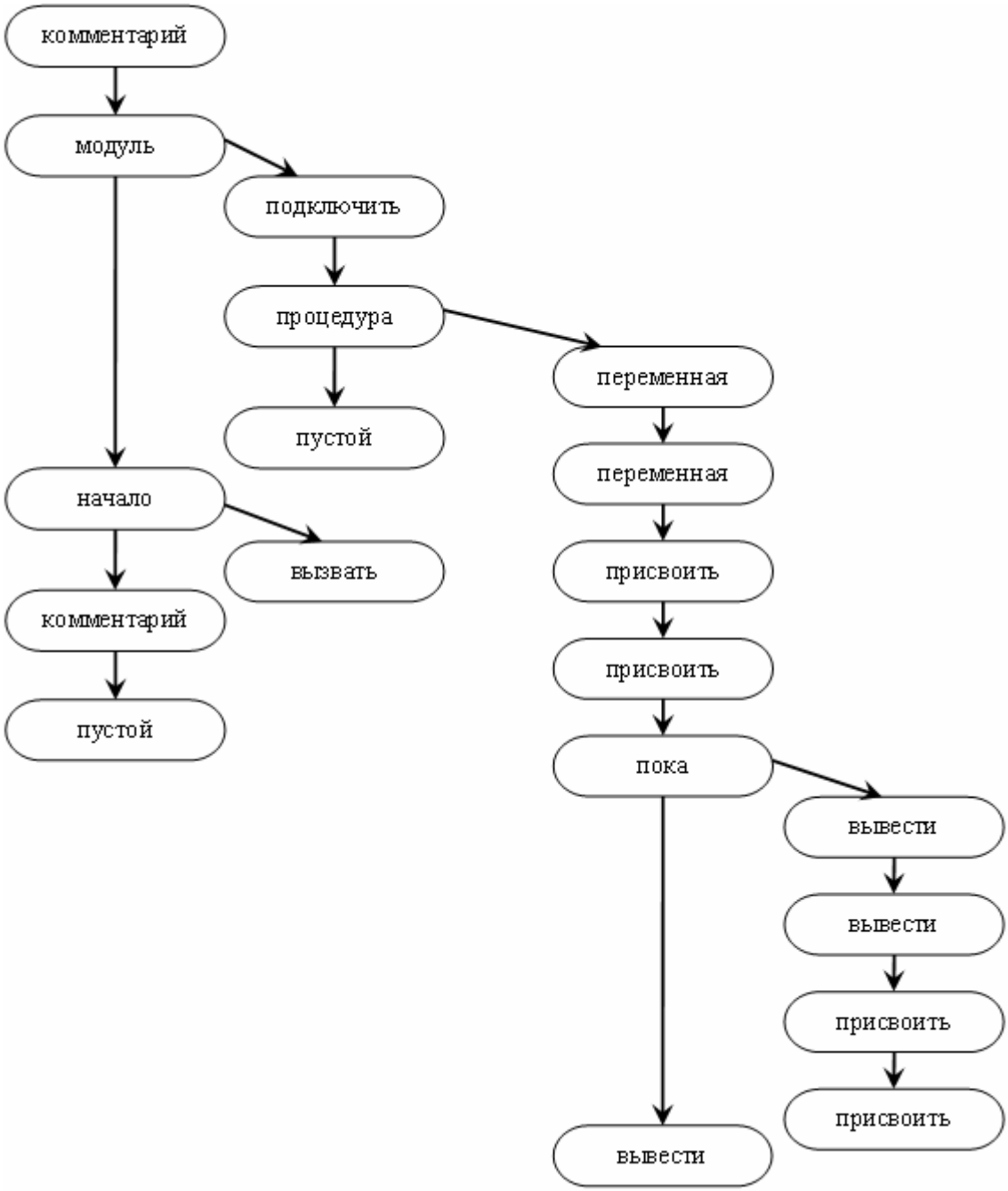

Рис. 2 – Семантическое дерево программы Факториал

В-третьих, в виде семантического дерева может быть представлен любой документ, состоящий только из узлов-комментариев. Система позволяет включать в комментарий текст, рисунки, таблицы – все, что позволяет включать формат RTF. Поэтому справочный и методический обучающий материал можно создавать непосредственно в редакторе кода.

И наконец, явное отделение внутреннего представления программы от внешнего позволило реализовать идею сменяемого синтаксиса во внешнем представлении [2,3]. Фундаментом реализации семантического дерева является паттерн Наблюдатель (Observer)

[4]. Как известно, в этом паттерне устанавливаются отношения между субъектом и наблюдателем. Субъект хранит у себя список наблюдателей, которых оповещает при каких-либо изменениях в своем состоянии.

В рамках данной реализации субъект наблюдения реализуется абстрактным классом SemanticSubject, который является корнем обширной иерархии классов, часть которой показана на рис. 3. Символом <<A>> помечены абстрактные классы. В этой иерархии определены все классы-операторы, которые, как и операторы учебного языка, делятся на два вида: однострочные (простые) и многострочные (блочные). Помимо операторов в иерархию семантических объектов включены элементы операторов, которые сами имеют сложную структуру: выражения, список параметров процедур-функций-методов, не элементарные типы. Как правило, эти элементы операторов разрешено редактировать в составе оператора.

Семантическое дерево, элементами которого являются объекты-операторы, реализуется отдельным классом SemanticTree. В абстрактном классе SemanticOperator (см. на рис. 3 класс Оператор) определены все необходимые поля-ссылки.

Множество классов-наблюдателей, составляющих иерархию классов-блоков с базовым абстрактным классом Block, реализуют интерфейс ISemanticObserver. Все эти классы так или иначе связаны с отображением семантического дерева в окне редактора. Сам редактор, реализуемый классам Editor, тоже является наблюдателем. Паттерн Наблюдатель используется еще в некоторых случаях, в частности, для наблюдения за курсором.

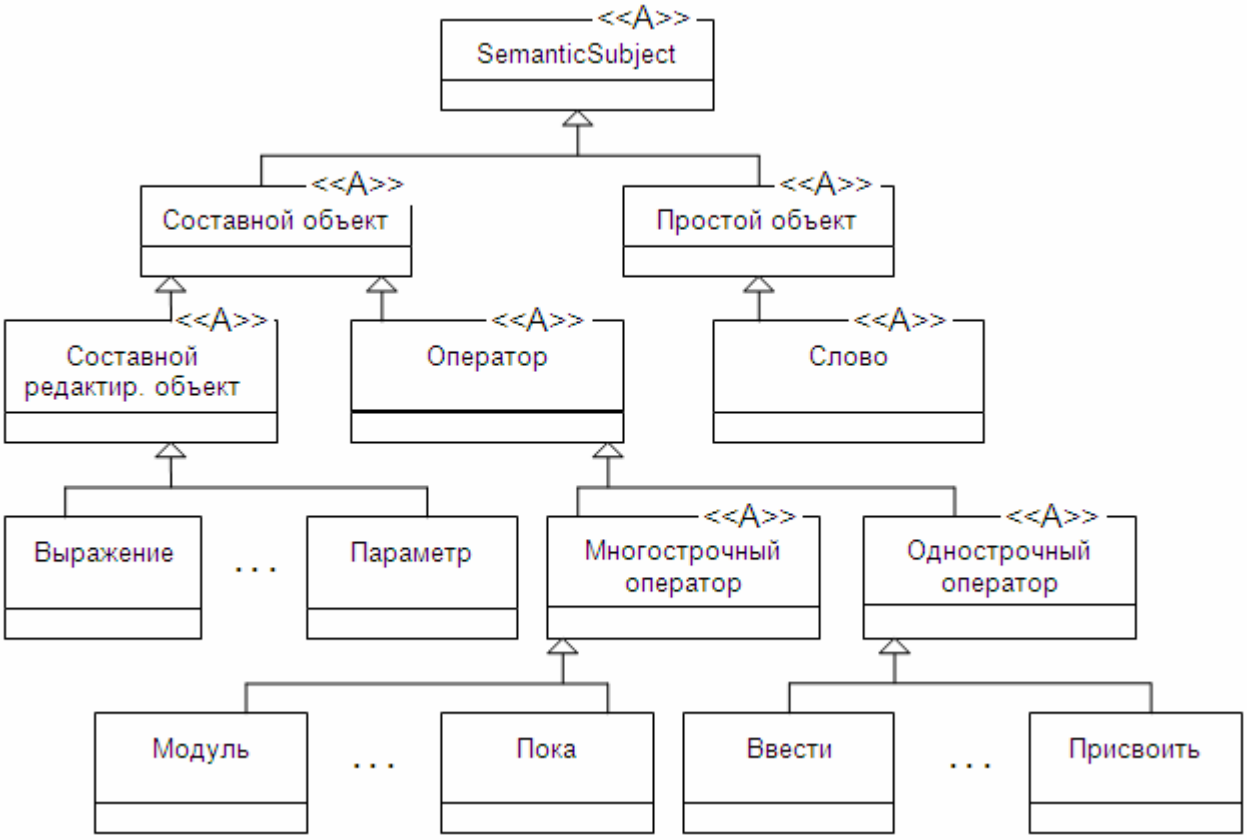

Рис. 3 – Иерархия классов субъекта наблюдения

Помимо паттерна Наблюдатель в реализации широко применяется паттерн Команда [4] (Command). В Semantic IDE определен классический базовый абстрактный класс Command с методами Execute() и Undo(). Посредством паттерна Компоновщик (Composite) реализован классический класс MacroCommand, определяющий составную команду. Такой подход позволил определить наборы базовых команд для различных объектов модели, а уже из них компоновать более сложные составные команды. Например, базовые команды определены для выражения, списка параметров, модуля, семантического дерева, слова и т.п. В системе реализован класс CommandHistory для хранения истории выполненных команд.

Команды IDE составляют отдельную иерархию с базовым абстрактным классом IDECommand, но тоже скомпонованы по «тематическим» группам. Например, все команды для работы с проектом собраны в одну группу. Для хранения команд IDE реализован классический стек команд с методами Undo() и Redo().

Описанный подход с широким использованием паттернов позволил существенным образом снизить затраты времени и сил при модификации и развитии системы. Построение семантического дерева программы редактором позволяет полностью отказаться от фазы анализа (как лексического, так и синтаксического) при реализации конвертеров учебного языка в промышленные языки – это также существенным образом упрощает разработку.

Литература

1. Грачёв А.Д., Лаптев В.В. Разработка учебного языка программирования и интерпретатора для обучающей среды // Объектные системы-2012: материалы VI Международной научно-практической конференции (Ростов-на-Дону, 10-12 мая 2012г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ ЮРГТУ (НПИ), 2012. – 110 с., с. 92-101.
2. Грачёв А.Д., Лаптев В.В. Интегрированная среда для обучения программированию // Объектные системы – 2013: Материалы VII Международной научно-практической конференции (Ростов-на-Дону, 10-12 мая 2013 г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ (ф) ЮРГТУ (НПИ), 2013. – 118 с., с.17-24
3. Грачёв А.Д., Лаптев В.В. Семантическая интегрированная среда для обучения программированию // Педагогическая информатика. – 2013, №2. – с. 71-81.
4. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – Спб.: Питер, 2001. – 368 с.

Шахтинский институт (филиал)
федерального государственного бюджетного образовательного учреждения высшего профессионального образования "Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова"

Донской государственный технический университет
Институт сферы обслуживания и предпринимательства (филиал) ДГТУ в г.Шахты

IX Международная научно-практическая конференция

Подробную информацию о конференции можно найти на официальном сайте www.objectsystems.ru

ОБЪЕКТНЫЕ СИСТЕМЫ – 2015

(с изданием сборника материалов конференции)

Информационные партнёры конференции:

Сообщество системных аналитиков

Теоретический и прикладной научно-технический журнал
"**Информационные технологии**"
с ежемесячным приложением

Ежемесячный научно-технический и производственный журнал
"**Автоматизация в промышленности**"

Журнал "**Информационные технологии и вычислительные системы**"

10–12 мая 2015 г , Россия, Ростов-на-Дону

Статьи сборников конференции индексируются в РИНЦ
http://elibrary.ru/title_about.asp?id=48777

Конференция посвящена принципам проектирования, реализации и сопровождения объектных систем и включает в себя обсуждение широкого круга проблем. В организации и работе конференции принимают участие как известные учёные, так и крупные специалисты в области разработки корпоративных информационных систем, представители ВУЗов и коммерческих организаций России, ближнего и дальнего зарубежья (Греции, Польши, Испании). Конференция является заочной. По окончании работы конференции публикуется

сборник научных трудов авторов с присвоением ему ISBN-кода, который будет разослан в крупнейшие научные библиотеки России. Электронная версия сборника размещается в ведущих информационных каталогах и доступна на сайте конференции. Авторам каждого доклада высылается печатная версия сборника материалов конференции и именные сертификаты. Лучшие, по мнению рецензентов, доклады будут рекомендованы (после соответствующей доработки) к бесплатной публикации в журналах "Автоматизация в промышленности", "Информационные технологии" и "Информационные технологии и вычислительные системы", входящих в перечень ведущих рецензируемых научных журналов и изданий, рекомендованных ВАК. Лучший доклад по UML-моделированию награждается книгой Иванова Д. Ю. и Новикова Ф.А. "Моделирование на UML. Теория, практика, видеокурс" (www.umlmanual.ru) с автографами авторов. Автор лучшего доклада, посвящённого методам преподавания объектных технологий в ВУЗе, получает право бесплатной публикации одного доклада сходной тематики на следующей конференции.

Оргкомитет конференции

1. Прокопенко Николай Николаевич, д.т.н., проф., Заместитель директора по научно-исследовательской работе, Институт сферы обслуживания и предпринимательства (филиал) ДГТУ в г.Шахты, Россия, Шахты ***(председатель конференции)***
2. Олейник Павел Петрович, к.т.н., Системный архитектор программного обеспечения, Астон, Россия, Ростов-на-Дону ***(сопредседатель конференции)***
3. Божич Владимир Иванович, д.т.н., проф., Кафедра "Информационные системы и радиотехника", Институт сферы обслуживания и предпринимательства (филиал) ДГТУ в г.Шахты, Россия, Шахты
4. Сидельников Владимир Иванович, д.т.н., проф., Заведующий кафедрой "Экономика и прикладная математика", Педагогический Институт Южного Федерального университета, Россия, Ростов-на-Дону
5. Черкесова Эльвира Юрьевна, д.э.н., проф., Заведующая кафедрой "Управление персоналом, инновациями и качеством", Шахтинский институт (филиал), Южно-Российский государственный технический университет (Новочеркасский политехнический институт), Россия, Шахты
6. Михайлов Анатолий Александрович, д.т.н., проф., Кафедра "Автоматизированные системы управления", Южно-Российский государственный технический университет (Новочеркасский политехнический институт), Россия, Новочеркасск
7. Кравчик Вячеслав Георгиевич, к.т.н., доц., Кафедра "Энергетика и БЖД", Институт сферы обслуживания и предпринимательства (филиал) ДГТУ в г.Шахты, Россия, Шахты

Международный программный комитет конференции

1. Euclid Keramopoulos, Ph.D., Lecturer, Alexander Technological Educational Institute of Thessaloniki, Греция, Салоники
2. Piotr Habela, Ph.D., Assistant Professor, Polish-Japanese Institute of Information Technology, Польша, Варшава
3. Erki Eessaar, Ph.D., Associate Professor, Acting Head of Chair, Faculty of Information Technology: Department of Informatic, Tallinn University of Technology, Эстония, Таллин
4. German Viscuso, MSc in Computer Science, Marketing, Versant Corp., Испания, Мадрид
5. Кузнецов Сергей Дмитриевич, д.т.н., проф., Факультет вычислительной математики и кибернетики, МГУ имени М. В. Ломоносова, Главный научный сотрудник Института системного программирования РАН, член ACM, ACM SIGMOD и IEEE Computer Society, Россия, Москва
6. Шалыто Анатолий Абрамович, д.т.н., проф., лауреат премии Правительства РФ в области образования, Заведующий кафедрой "Технологии программирования", Санкт-Петербургский государственный университет информационных технологий механики и оптики, Россия, Санкт-Петербург
7. Кирютенко Александр Юрьевич, к.ф.-м.н., Директор по ИТ, Астон, Россия, Ростов-на-Дону
8. Галиаскаров Эдуард Геннадьевич, к.х.н, доц., Ивановский государственный химико-технологический университет, Россия, Иваново
9. Чекирис Александр Владимирович, Начальник отдела технического проектирования и НСИ, НИИЭВМсервис, Беларусь, Минск
10. Векленко Ирина Юрьевна, к.э.н., Ведущий системный аналитик, РИТКОН, Россия, Черноголовка

11. Малышко Виктор Васильевич, к.ф.-м.н., доц., Факультет вычислительной математики и кибернетики, МГУ имени М. В. Ломоносова, Россия, Москва
12. Жилякова Людмила Юрьевна, к.ф.-м.н., с.н.с., Институт проблем управления им. В.А. Трапезникова РАН, Россия, Москва
13. Шахгельдян Карина Иосифовна, д.т.н., Начальник информационно-технического обеспечения, Владивостокский государственный университет экономики и сервиса, Россия, Владивосток
14. Добряк Павел Вадимович, к.т.н., доц., Уральский государственный технический университет, Россия, Екатеринбург
15. Байкин Александр Сергеевич, Ведущий системный аналитик, Автомир, Россия, Москва
16. Аверин Алексей Иванович, Системный аналитик, Астон, Россия, Ростов-на-Дону
17. Лаптев Валерий Викторович, к.т.н., доц., Кафедра "Автоматизированные системы обработки информации и управления", Астраханский государственный технический университет, Россия, Астрахань
18. Ермаков Илья Евгеньевич, Генеральный директор, ООО "Метасистемы", Россия, Орёл
19. Иванов Денис Юрьевич, Консультант, Ай Ти Консалтинг, Россия, Санкт-Петербург

Основные секции конференции

1. Графические нотации, используемые при объектном проектировании ИС
2. Принципы объектного проектирования информационных систем
3. Инструменты объектного моделирования
4. Теория объектно-ориентированного программирования
5. Методы (шаблоны) объектно-реляционного отображения
6. Реализация и использование объектных расширений в реляционных СУБД
7. Проектирование, разработка и реализация распределённых систем
8. Типовые реализации КИС с применением объектных технологий
9. Принципы организации и реализации объектных баз данных
10. Проблемы реализации объектных СУБД
11. Имитационное моделирование объектных систем
12. Темпоральные объектные системы
13. Проблемы изучения (преподавания) объектных технологий в ВУЗе

Ключевые даты конференции

01.01.2015 – 15.04.2015 – приём заявок на участие в конференции и докладов к публикации
16.04.2015 – 30.04.2015 – рецензирование и корректировка присланных докладов
01.05.2015 – 09.05.2015 – рассылка уведомлений о принятии докладов на конференцию и приём квитанций об оплате оргвзноса
10–12 мая 2015 – формирование сборника трудов и сертификатов
01.07.2015 – 15.07.2015 – размещение на сайте конференции электронного макета сборника материалов с присвоенными ISBN-кодами и именных сертификатов, подтверждающих участие в конференции
II – III квартал 2015 г. – рассылка печатной версии сборника материалов авторам, и в научные библиотеки РФ. Рассылка бумажных именных сертификатов. Регистрация электронной версии в специализированных каталогах и РИНЦ

Подробную информацию о конференции можно найти на официальном сайте www.objectsystems.ru

Для заметок / Notes

Для заметок / Notes

Для заметок / Notes

Для заметок / Notes

Отпечатано 01.08.2014

Верстка – **Галиаскаров Э.Г.**
Дизайн обложки – **Олейник П.П.**
Корректоры – **Лаптев В.В., Ермаков И.Е.**
Ответственный за выпуск – **Галиаскаров Э.Г.**
Подписано в печать **25.07.2014**. Формат А4
Бумага офсетная. Печать цифровая
Усл.печ.л. **13,7**
Тираж 300 экз
Заказ № 987

ISSN 2309-8856
9 772309 885145
01
ISBN 978-5-9903782-7-8
9 785990 378278
ISBN 978-5-9903782-8-5
9 785990 378285