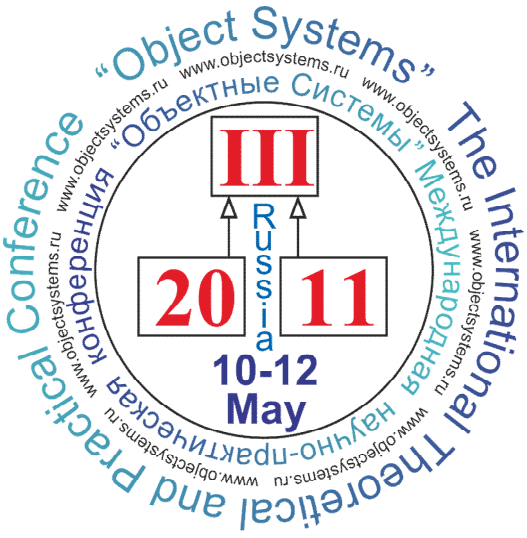
Шахтинский институт (филиал)
государственного образовательного учреждения
высшего профессионального образования
«Южно-Российский государственный технический университет
(Новочеркасский политехнический институт)»

Южно-Российский государственный университет
экономики и сервиса



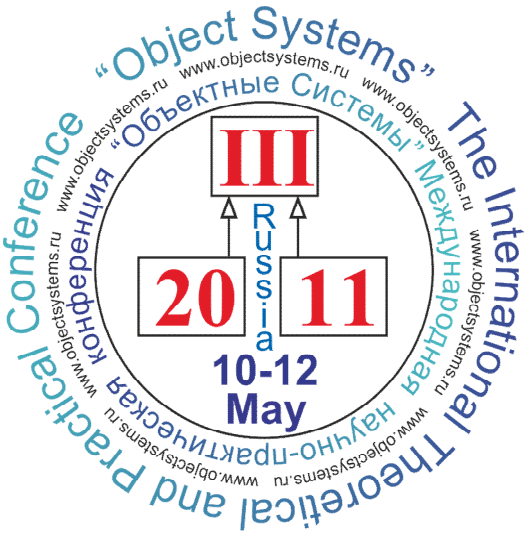
ОБЪЕКТНЫЕ СИСТЕМЫ – 2011

Материалы III Международной научно-практической конференции

Россия, Ростов-на-Дону
10-12 мая 2011 г.

Шахтинский институт (филиал)
государственного образовательного учреждения
высшего профессионального образования
«Южно-Российский государственный технический университет
(Новочеркасский политехнический институт)»

Южно-Российский государственный университет
экономики и сервиса



ОБЪЕКТНЫЕ СИСТЕМЫ – 2011

Материалы III Международной научно-практической конференции

Россия, Ростов-на-Дону
10-12 мая 2011 г.

УДК 004 (05)
ББК 32.97-018
О-29

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. – 160 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) государственного образовательного учреждения высшего профессионального образования «Южно-Российский государственный технический университет (Новочеркасский политехнический институт)»

Южно-Российский государственный университет экономики и сервиса

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., Ректор Южно-Российского государственного университета экономики и сервиса, 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", Южно-Российский государственный университет экономики и сервиса, 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 "Информационные технологии и управление", 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 Владислав Георгиевич, к.т.н., доц., Кафедра "Энергетика и БЖД", Южно-Российский государственный университет экономики и сервиса, 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., кафедра "Экономика и прикладная математика", Педагогический Институт Южного Федерального университета, Россия, Ростов-на-Дону
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, Технологический институт им. Н.Н. Поликарпова "Государственный университет – учебно-научно-производственный комплекс", Технический директор, НПО "Тесла", Россия, Орёл
19. Иванов Денис Юрьевич, Консультант, Ай Ти Консалтинг, Россия, Санкт-Петербург

Рецензенты

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

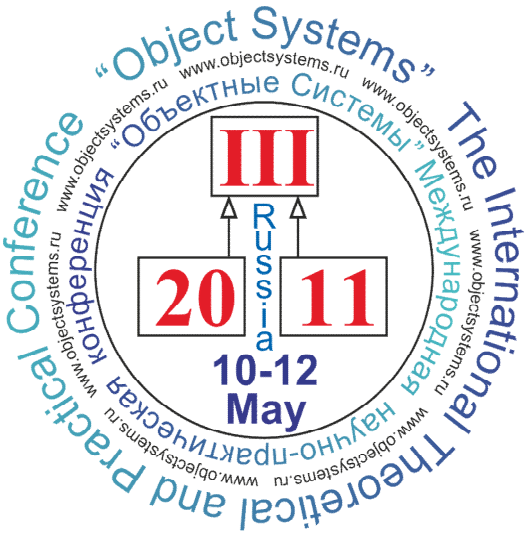
Редакторы

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

ISBN 978-5-9902226-5-6

Shakhty Institute (Branch) of South Russian State Technical University (Novocherkassk Polytechnic Institute)

South Russian State University of Economics and Service



Object Systems – 2011

Proceedings of the Third International Theoretical and Practical Conference

Edited by Pavel P. Oleynik

Rostov-on-Don, Russia
10-12 May 2011

Object Systems – 2011: Proceedings of the Third International Theoretical and Practical Conference. Rostov-on-Don, Russia, 10-12 May, 2011. Edited by Pavel P. Oleynik

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 South Russian State Technical University (Novocherkassk Polytechnic Institute)

South Russian State University of Economics and Service

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, Rector, South Russian State University of Economics and Service, 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, South Russian State University of Economics and Service, 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 Information Technologies and Management, 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, South Russian State University of Economics and Service, 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., Associated Professor of Economics and Applied Mathematics, Southern Federal University, Pedagogical Institute, Russia, Rostov-on-Don
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, Lecturer, Polycarpov Institute of Technology at the State University – Educational-Scientific-Industrial Center, CTO, NPO "Tesla", Russia, Orel
19. Denis Yu. Ivanov, Consultant, IT Consulting, Russia, Saint-Petersburg

Reviewers

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

Editors

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

ISBN 978-5-9902226-6-3

Содержание / Contents

---

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

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

[3] Лауреат номинации "**Лучший доклад о методах преподавания объектных технологий в ВУЗе**". Автор доклада награждается правом бесплатной публикации одного доклада по данной тематике на следующей конференции

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

[5] Лауреат номинации "**Лучший доклад по UML-моделированию**". Автор доклада награждается книгой Иванова Д. Ю. и Новикова Ф.А. "**Моделирование на UML. Теория, практика, видеокурс**" (www.umlmanual.ru) с автографами авторов

Содержание / Contents

[1] Лауреат номинации "**Лучший доклад о методах преподавания объектных технологий в ВУЗе**". Автор доклада награждается правом бесплатной публикации одного доклада по данной тематике на следующей конференции

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

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

[4] Статья рекомендована к опубликованию в журнале "**Автоматизация в промышленности**"

[5] Статья рекомендована к опубликованию в журнале "**Автоматизация в промышленности**"

УДК 004.432

ЭВОЛЮЦИЯ ИДЕЙ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ ТЕХНОЛОГИЙ[1]

Добряк Павел Вадимович, к.т.н., Уральский Федеральный Университет,
Россия, Екатеринбург, goodsoul@mail.ru

Объектно-ориентированные технологии лидируют сегодня при разработке программного обеспечения. Большинство технологий являются в той или иной степени объектно-ориентированными, их создатели часто заявляют об «объектной ориентации» в рекламных целях. Полезно проанализировать технологии не с той точки зрения, в какой степени они являются объектно-ориентированными, а чтобы проследить, как развиваются в них идеи объектно-ориентированной парадигмы; показать, что под объектно-ориентированными языками скрывается несколько, вообще говоря, разных стилей программирования; обратить внимание на идеи, которые находятся пока на обочине без реализации в лидирующем семействе языков C++, Java, C#, но могут быть востребованными; и сделать прогнозы.

Принципы ООП сформулированы достаточно обобщенно, чтобы под них подходило большое количество технологий, в то же время идеи технологий могут эволюционировать со временем, вполне оставаясь в рамках объектно-ориентированного подхода. Это одна из причин успеха ООП. Формулировки и само количество принципов может отличаться. Основными принципами являются:

1. абстракция – выделение значимых свойств и опущение незначимых, использование основного понятия – объект;
2. инкапсуляция – совместное определение данных и кода внутри одной языковой конструкции;
3. полиморфизм – способность одного и того же кода описывать объекты разной природы.

К дополнительным принципам можно отнести:

1. наследование (фактически – способность одного описания пользоваться другим как самим собой без повторного кодирования; тесно связано с полиморфизмом);
2. агрегация (вложенные объекты).

Посмотрим, как эти принципы приводят к разным стилям программирования, проанализировав программы.

Изначально программа на объектно-ориентированном языке должна была представлять собой *множество объектов, обменивающихся сообщениями*. То есть классы, вообще говоря, необязательны (иногда такие «бесклассовые» технологии называют не объектно-ориентированными, а объектными). Уже на Smalltalk [1] обмен сообщениями маскируется под вызов методов. Программа на C++ представляет собой *объекты – экземпляры классов, организованных в иерархию наследования, вызывающие друг у друга свойства и методы* (уже довольно далеко от «объектов, обменивающихся сообщениями»!).

Объектно-ориентированные языки из экзотики стали обыденностью при появлении графического пользовательского интерфейса. Элементы оконного интерфейса (кнопки, формы и прочее) очень удобно описывать в виде объектов со свойствами и методами. При этом ООП поглотило другую парадигму – событийную (программисты, переходящие с C++ Builder на Visual C++ с удивлением видели в своем коде таблицы событий, которые в Visual C++ открыты для программиста, и учились не обращать на них внимание). События и сообщения – родственные механизмы. Только, если у сообщения есть адресат, то событие генерируется «в эфир» и у него есть подписчики (которых может быть много), которые реагируют на это событие. Событие само может быть описано как объект. Изменяется само понятие объекта как совокупности свойств и методов. Заодно добавляются внутренние

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

классы – классы, вложенные внутрь других классов (у которых в C# есть модификаторы доступа). Теперь объект – это свойства, методы, внутренние классы и события (полноценно реализованные в C#). Программа представляет собой *объекты – экземпляры классов, организованных в иерархию наследования, вызывающие друг у друга свойства и методы, генерирующие и подписывающиеся на события.*

Распределенные приложения и технологии повторного использования кода также обогатили идеями ООП. По аналогии с компьютерами, построенными по магистрально-модульному принципу, когда различные компоненты взаимозаменяемы и могут соединяться в разных конфигурациях, в ООП пришло понятие интерфейса – класса без полей, целиком состоящего из абстрактных (не реализованных) методов. Объекты могут взаимодействовать, если есть интерфейсы, который они реализуют. То есть класс, к экземпляру которого обращается другой объект, наследует интерфейс, обязательно определяя каждый метод интерфейса. Программа представляет собой *классы, организованные в иерархию наследования и реализующие интерфейсы, экземпляры которых вызывают друг у друга свойства и методы, генерируют и подписываются на события*. При этом нужно обратить внимание на то, что принцип инкапсуляции несколько нарушается. Если мы до этого данные и код, который их обрабатывает, описывали вместе внутри класса, то теперь, группируя методы в интерфейс, мы удаляем методы от класса на «некоторое расстояние». Назовем эту проблему «вечным дуализмом данных и кода». С одной стороны, у программистов есть стремление для усиления ясности и простоты отделить данные и код, с другой стороны, есть потребность в манипулировании кодом, как данными (например, в эволюционных алгоритмах). В Lisp [2] переменных нет вообще, в то же время есть обозначения. К этой проблеме мы еще вернемся.

Такие технологии, как CORBA и ActiveX [3], также изменили понятие объекта. Если рассматривать DLL (предка ActiveX) как множество функций, откомпилированных и повторно используемых, то откомпилированный и повторно используемый объект – компонент – делает программирование более изощренным. Компонент может быть активным, он сам может вызывать что-нибудь из контейнера. Значит, необходимы события. Для удобства программист может добавить свойства и методы откомпилированному компоненту (расширенные свойства и методы). Фактически объявленные внутри контейнера, расширенные свойства и методы вызываются у компонента, как если бы они были настоящими свойствами и методами компонента. Кроме того, компонент может потребовать, чтобы контейнер предоставил ему доступ к ряду свойств контейнера, называемых свойствами окружения (например, когда компонент хочет подстроиться по цвету и шрифту к контейнеру). Расширенные методы появляются в C# для несколько другой задачи – более короткой записи вызова цепочки методов, что затем применяется в LINQ [4]. Если в C# добавить слово this у аргумента функции в её сигнатуре, то мы сможем у объекта класса, к которому принадлежит аргумент, вызвать этот метод, как будто это собственный метод этого класса. Получается, что метод как будто «отсоединился» от класса и стал принадлежать нескольким классам. Это еще один случай «вечного дуализма данных и кода» (похожая ситуация в языке CLOS [2], там классы и методы понимаются не так, как в языке C++, метод – отдельная от класса сущность).

Расширенные свойства и методы заставляют нас задуматься о том, что такое «экземпляр класса». Если до этого объект создавался на основе комбинации класса экземпляра и его родительских классов, то теперь к нему присоединяются еще свойства и методы других классов – контейнеров. Получается, что есть объекты, которые части друг друга считают своими – нечеткие объекты (по аналогии с нечеткими логиками, множествами и алгоритмами). Программа представляет собой *классы и объекты. Классы организуются в иерархию наследования и реализуют интерфейсы. Объекты могут формироваться на основе нескольких классов, входить внутрь друг друга или иметь общие части. Объекты вызывают друг у друга свойства и методы, генерируют и подписываются на события*.

90-е годы стали временем интеграции трех ответвлений отрасли ИТ: программирования на языках ООП, баз данных и систем искусственного интеллекта. Эта интеграция не закончилась до сих пор, порождая множество идей. Так, например, объектное расширение Lisp – CLOS, содержит идеи, отсутствующие пока в языках семейства C++ (рассмотрим их ниже). В реляционные базы данных добавились методы, связи наследования и агрегации (получились объектно-реляционные базы данных). В языки программирования добавились перманентность (персистентность) объектов [5] – способность объекта сохранять свое состояние между запусками программы, а впоследствии транзакции и триггеры (получились объектно-ориентированные базы данных). Разновидностью триггеров можно считать конструкторы и деструкторы.

Модель фрейм-слот появилась раньше ООП [6]. Прослеживаются четкие аналогии между фреймами и классами. Фрейму соответствует класс, слоту – свойство и метод, при расширении триггерами в объектно-ориентированных базах данных (действиями, выполняемыми до или после создания, изменение и удаления объектов), фацетам также находится место в ООП (демонам соответствуют триггеры, значения по умолчанию есть во всех моделях).

В языке C++ мы говорим о двух видах виртуальности: виртуальных функциях (если указатель на экземпляр класса-предка на самом деле указывает на экземпляр класса-потомка, то через этот указатель вызывается метод класса-потомка, если метод класса-предка объявлен виртуальным) и виртуальном наследовании (исключает множественное создание экземпляра класса – общего предка нескольких классов, от которых идет множественное наследование класса-потомка). По аналогии с виртуальностью в C++ можно сказать, что в CLOS есть еще один вид виртуальности – виртуальные аргументы. Допустим, что мы вызвали функцию, где в качестве аргумента передали не описанный в сигнатуре экземпляр класса, а экземпляр класса-потомка. Если мы хотим, чтобы функция работала с ним как с классом – потомком, нужно, чтобы этот аргумент был виртуальным.

До сих пор мы определяли объект через его составные части – свойства, методы, события и пр. Можно взглянуть на объекты по их разновидностям. Часто в литературе по C++ написано, что объект – это экземпляр класса или класс. Является ли класс полноценным объектом? Нет, во всяком случае, не в языках семейства C++. Проведем анализ на основе C++. У класса нет свойств и методов (хотя статические свойства и методы можно считать свойствами и методами класса). Не существует полноценных указателей на класс, хотя много суррогатов указателей – шаблоны и идентификаторы класса в ActiveX (можно спорить, нужны ли вообще указатели и их заменители). Нет операций над классами, хотя множественное наследование можно рассматривать как сумму классов-предков. У класса нет своего класса, к которому он мог бы принадлежать (то есть метакласса). Служебные метаклассы есть в Smalltalk. Полноценные многоуровневые метаклассы, создаваемые программистом, есть в языке CLOS.

Полноценные метаклассы переносят нас из двумерного мышления (наследование, агрегация) к трехмерному (наследование, агрегация, метаклассы). Это более трудное мышление. Может быть, поэтому метаклассы плохо приживаются в языках программирования. Пример трехмерного мышления: бакалавр -> студент -> человек (наследование), студент -> группа -> факультет (агрегация), человек -> понятие ->слово (метаклассы). Программа будет представлять собой *объекты, объединенные в три иерархии: наследование (в том числе с реализацией интерфейсов), агрегацию и метаклассы. Объекты могут формироваться на основе нескольких описаний, входить внутрь друг друга или иметь общие части. Объекты вызывают друг у друга свойства и методы, генерируют и подписываются на события*. Проанализируем эти иерархии.

Под наследованием можно понимать несколько различных идей: дополнение класса-предка новыми свойствами и методами класса-потомка, переопределение свойств и методов предка в потомке, использование потомком свойств и методов предка как своих собственных, компоновка методов по веточке наследования. Полноценной компоновки в

языках семейства C++ нет. В языке CLOS у методов стоят модификаторы before, after и around – куда поставить метод: в начало, конец или даже вокруг. Результирующий метод собирается из всех методов по цепочке наследования в зависимости от модификаторов расположения.

До сих пор идут споры, нужно ли полноценное множественное наследование [7,8] (разработчики Java и C# убрали полноценное множественное наследование). Противники говорят о том, что множественное наследование на практике порождает большое количество проблем при совпадении имен, сторонники – что есть предметные области, особенно в системах искусственного интеллекта, когда без множественного наследования не обойтись. Но можно придумать и более сложные виды наследования. Использование и переопределение свойств и методов не обязательно должно быть иерархическим, теоретически, оно может быть и циклическим.

Бывают случаи, когда мы не можем четко сказать, использовать наследование или агрегацию. Это происходит тогда, когда часть подобна целому. Например, административно-территориальное деление: страна состоит из областей, но структурно области подобны стране (столица, площадь, количество населения – свойства те же). Особая семантика для гибридных связей наследования и агрегации есть в проекте Orion [5].

Может быть, есть смысл помимо различных связей наследования, агрегации и метаклассов, создавать и пользовательские связи, определяемые программистом? В этом случае ООП поглотит парадигму семантических сетей. Это в урезанном виде сделано в средствах объектно-ориентированного проектирования. Поскольку UML предназначен не только для разработки структуры объектно-ориентированных программ, но и баз данных, классы могут соединять сеть ассоциативных связей, не обязательно бинарных. Программа будет представлять собой *сеть объектов, соединенных различными связями, в том числе – определяемых программистами. Объекты могут формироваться на основе нескольких описаний, входить внутрь друг друга или иметь общие части. Объекты вызывают друг у друга свойства и методы, генерируют и подписываются на события*.

Интерпретаторы и виртуальные машины позволяют разработчикам языков делать программы более гибкими – изменять структуру классов во время выполнения программы. На C# можно добавлять в класс новые свойства и методы или создавать новый класс во время выполнения программы. Для реализации функционального программирования в C# введены деревья решений (формулы можно представить в виде объектов, «замораживать» их выполнение, изменять и компилировать – пример «вечного дуализма данных и кода»). Язык тем более объектно-ориентирован, чем большее количество элементов представляют собой объекты. В языке Smalltalk, считающимся наиболее чистым объектно-ориентированным языком, структурные операторы являются методами у служебных классов, например, оператор условия – метод у объекта истина. Можно сформулировать идею, что программа и её части – это тоже объекты. Таким образом, объектами будут классы и их экземпляры, события, связи между классами и части программы.

В 21 веке понятия класса разработчикам информационных систем стало явно не хватать. Группа разработчиков, называемая «бандой четырех», выявила стандартные ситуации при объектно-ориентированном проектировании – паттерны проектирования – конфигурации классов, часто встречающиеся при разработке [9]. Впоследствии ряд паттернов был встроен в языки на уровне синтаксических конструкций. Например, паттерн синглтон (требование единственности экземпляра класса), был встроен в C#. Развитием концепций сообщений и событий являются паттерны «команда», «наблюдатель» и «цепочка обязанностей», можно предположить, что в будущем появятся языковые конструкции, их реализующие.

В языке BETA [10] есть интересная конструкция, являющаяся обобщением понятий «класс» и «функция», также названная паттерном (еще один пример «вечного дуализма данных и кода»). Программисты на языках семейства C++ привыкли к различиям между функциями и классами, не замечая, что между этими понятиями сходства больше, чем

различий. Так, статические поля функции можно считать свойствами, вызову функции соответствует вызов конструктора класса. Осталось только добавить полям функции модификаторы доступа и экземпляризацию функции, и получится класс. Между паттернами объектно-ориентированного проектирования и паттернами языка BETA есть родство, если паттерны проектирования рассматривать не как стандартные ситуации, а как конфигурации классов. Интересно то, что, находясь в объектно-ориентированной парадигме, мы отошли от самого понятия «объект» и перешли к понятию паттерн.

Часто высказывается точка зрения, что объектно-ориентированные технологии лидируют потому, что ООП наиболее соответствует мышлению человека. Отчасти конструкции ООП соответствуют грамматическим структурам индоевропейских языков. На этом основываются методы объектно-ориентированного анализа. Подлежащему можно поставить в соответствие объект, сказуемому – метод, определениям – свойства, дополнениям и обстоятельствам – аргументы метода. Паттерны могут описывать более сложные конструкции – придаточные предложения, причастные и деепричастные обороты. Возникает парадокс: близость к естественному языку удобна для проектирования, но это также приближает компьютерный язык к естественному, что делает тяжелым его изучение.

Если к предыдущим идеям добавить паттерны, то *программы будут представлять собой динамические сети паттернов (формирующихся на основе нескольких описаний) внутри других сетей, обменивающихся сообщениями и генерирущих события (тоже считающиеся паттернами), при этом код программы также представляет собой паттерны.* Это программирование, в котором иерархии заменяются сетями, что приводит к совсем другому взгляду на программы [11]. Этот переход (от иерархий к сетям) соответствует сдвигу парадигм в разных отраслях науки, например – в экологии, экономике и квантовой физике [12].

Когда мы говорим об объектно-ориентированных языках, нужно обязательно обратить внимание на то, что распространенные языки являются мультипарадигменными, что не удивительно (например, формулы удобно писать на математическом языке, а не предикатами Пролога). При распространении ООП объектные надстройки появились над самыми разными языками, например, над Lisp. В C++ можно выделить четыре уровня: математические и логические формулы, структурное программирование, функции, классы. По всей видимости, объектно-ориентированная надстройка не всегда будет верхним уровнем. Уже сейчас при программировании Интернет-страничек код на языках высокого уровня обрамляется тегами. В C# встраивается функциональное программирование и декларативное программирование LINQ. «Матрешка» вложенных друг в друга парадигм заменяется перетеканием одной парадигмы в другую. Конструкции одного стиля встраиваются внутрь другого. Совсем не обязательно, что ООП сохранит свое «привилегированное» положение верхнего уровня в мультипарадигменном программировании.

Литература

1. Кирютенко Ю.А., Савельев В.А. Незнакомый Smalltalk. Ростовский государственный университет, 1997.
2. Jeff Dalton. A Brief Guide to CLOS. http://www.aiai.ed.ac.uk/~jeff/clos-guide.html
3. Деннинг А. ActiveX для профессионалов – СПб: Питер, 1998. – 624 с.: ил.
4. Трей Нэш. C# 2010. Ускоренный курс для профессионалов – Издательство: Вильямс, 2010 г.
5. Кузнецов С. Объектно-ориентированные базы данных – основные концепции, организация и управление: краткий обзор. http://www.citforum.ru/database/articles/art_24.shtml
6. М. Минский. Фреймы для представления знаний. М.: Мир, 1979.
7. Легалов А. О стрельбе по множественному наследованию. // Открытые системы №5-6, 2001 г.
8. Труб И. О проблемах множественного наследования. // Открытые системы №2, 2001 г.
9. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны проектирования. Издательство: Питер, 2007 г.
10. Ole Lehrmann Madsen, Birger Møller-Pedersen, Kristen Nygaard. Object-Oriented Programming in the BETA Programming Language. Addison-Wesley, June 1993, ISBN 0-201-62430-3, 350 pages

11. Люри Дж., Мансур Ш. Преодоление иерархий. //Открытые системы, 24.10.2001.
12. Добряк П.В., Калмыков А.А. Альтернативный подход по организации структур данных в объектно-ориентированных языках. Практика приборостроения №4[9], 2004. – с. 76 – 79.

УДК 004.4

ПРИМЕНЕНИЕ РАСПРЕДЕЛЕННЫХ СИСТЕМ УПРАВЛЕНИЯ В ПРОМЫШЛЕННОСТИ

Лучкин Николай Анатольевич, аспирант, Омский государственный технический университет, Россия, Омск, nikolay.luchkin@gmail.com

Эффективное управление предприятием в современных условиях невозможно без использования современных информационных технологий. Правильный выбор программного продукта и фирмы-разработчика – это первый и определяющий этап успешного решения автоматизации производства.

На сегодняшний день в области автоматизации промышленных предприятий выпускается все больше необходимого оборудования для проектирования распределенных систем управления (РСУ), которые внедряются на предприятиях разного направления вида производства. Первые РСУ были представлены на рынок в 1975 г. компаниями Honeywell (система TDC 2000) и Yokogawa (система CENTUM). Системы TDC-3000 были усовершенствованы и по настоящее время успешно эксплуатируются на предприятиях энергетической, металлургической, газовой промышленности по транспортировке и переработке газа, нефти и нефтепродуктов, целлюлозно-бумажной промышленности и др.

Основными современными на сегодняшний день распределенными системами управления являются следующие:

- ABB System 800xA
- CONTRONIC Hartmann-Braun
- Damatic XDi Valmet Automation
- Emerson DeltaV
- Honeywell Experion PKS
- Invensys Foxboro I/A Series
- Siemens SIMATIC PCS7
- Yokogawa CENTUM CS 3000

Современные распределенные системы управления эксплуатируются на базе:

- промышленных рабочих станций;
- многоканальных контроллеров;
- станций распределенного ввода/вывода;
- открытых промышленных сетей (Industrial Ethernet, Profibus, Modbus и др.);
- интеллектуальных устройств ввода/вывода;
- беспроводных устройств передачи информации;
- WEB-технологии передачи данными.

Архитектура распределенных систем управления определяется уровнями иерархии системы, такими как набор программно-технических средств, сетевая архитектура, возможности резервирования и другие необходимо важные параметры.

В зависимости от количества поставленных решаемых задач информационная емкость системы включает в себя различное число каналов ввода/вывода, способных обрабатывать от сотен до десятков тысяч сигналов. Для осуществления приема, хранения и обработки больших объемов параметров (информации) применяются базы данных реального времени (БДРВ) – реляционные, иерархические, объектно-ориентированные. Сервер БДРВ является ядром интегрированной системы управления, обеспечивая тем самым форматирование массивов данных, поступающих от технологического процесса для последующего их анализа и применения на верхнем уровне управления производством. БДРВ должна обеспечивать

синхронизацию, репликацию данных и обеспечивать резервирование для обеспечения отказоустойчивости в реальном масштабе времени.

Открытость системы управления предполагает применение в архитектуре системы открытых международных стандартов на аппаратное и программное обеспечение и т.п., допускающих совместное использование программно-аппаратных средств разных фирм-производителей.

Понятием "открытая система" можно считать определение, сформулированное комитетом IEEE POSIX 1003.0: "открытая система – это система, реализующая открытые спецификации на интерфейсы, сервисы и поддерживаемые форматы данных, достаточные для того, чтобы обеспечить должным образом разработанным приложениям возможность переноса с минимальными изменениями на широкий диапазон систем, совместной работы с другими приложениями на локальной и удаленных системах и взаимодействия с пользователями в стиле, облегчающем тем переход от системы к системе". Следовательно, можно сделать вывод, что система является открытой, если для нее определены и описаны используемые форматы данных и процедурный интерфейс, что позволяет подключить к ней "внешние", независимо разработанные компоненты.

Масштабируемость системы означает способность системы увеличивать свою производительность при добавлении ресурсов. Масштабируемость позволяет осуществлять модернизацию системы без критических изменений ее архитектуры. Масштабируемость – важный аспект электронных систем, программных комплексов, баз данных и т. п., если для них требуется возможность работать под большой нагрузкой.

Важными характеристиками системы управления являются: надежность; быстродействие; экономичность. Быстродействие системы определяется задействованным оборудованием «нижнего уровня», а также других узлов системы, и алгоритмическим обеспечением систем управления.

Основными техническими требованиями при проектировании распределенных систем управления являются:

- обеспечение широкого температурного диапазона работы технических средств локальных систем автоматического управления (САУ);
- распределенная система электропитания;
- обеспечение надежного контура заземлений на каждой отдельной площадке объекта автоматизации;
- защита контрольно-измерительных и информационных каналов от внешних воздействий, а также усиление передаваемых сигналов;
- выбор оптимального, с точки зрения эффективности, надежности и взаимозаменяемости составных частей, удовлетворяющего международным стандартам контроллерного оборудования;
- выбор оптимального, с точки зрения пылевлагонепроницаемости, а также защиты от электромагнитного излучения, коррозии и др. факторов, удовлетворяющего международным стандартам конструктива шкафа цехового контроллера, шкафов автоматики локальных САУ и автоматизированного рабочего места системного инженера (АРМ);
- обеспечение высоконадежных каналов обмена технологической информацией между отдельными автоматизированными объектами и централизованной системой управления и контроля;
- резервирование основной аппаратуры контроля и управления, а также наиболее важных каналов передачи информации;
- обеспечение аппаратного и программного аварийного останова технологического комплекса при аварийных ситуациях;
- обеспечение высокоэффективного человеко-машинного интерфейса в системе визуализации и мониторинга;

- обеспечение обмена данными по информационным каналам в реальном масштабе времени;
- эффективная, с точки зрения скорости обнаружения неисправности, и надежная диагностика программно-аппаратных средств;
- обеспечение обслуживающего персонала качественной эксплуатационной документацией, а также инструментом для монтажа и диагностики.

В данной статье рассматривается применение программных средств системы ТРЕЙС МОУД (TRACE MODE), разработка Российской фирмы AdAstra (г. Москва) для построения распределенной системы управления технологическим процессом. В пакете интегрирован набор инструментальных средств для создания как программного обеспечения (ПО) для РС контроллеров, так и ПО для рабочих станций. При этом имеется возможность создания систем управления с использованием контроллеров с встроенным ПО.

Базовая часть системы проектирования, которая называется «системой разработчика», содержит три программных инструмента:

- Редактор базы каналов (РБК) – инструмент разработки математической (функциональной) части системы в виде базы каналов и набора программ для обработки используемых переменных, а также создания системы обмена данными, системы архивирования и регистрации событий. РБК обеспечивает генерацией проектной документации;
- Редактор представления данных (РПД) – инструмент разработки экранных форм операторского интерфейса;
- Редактор шаблонов (РШ) – инструмент разработки шаблонов и сценариев формирования документации о ходе технологического процесса.

Кроме указанного система разработчика содержит набор встроенных драйверов для многих контроллеров, OPC и DDE клиенты и серверы, электронную справочную систему.

ТРЕЙС МОУД позволяет создавать многоуровневые, иерархически организованные, резервированные АСУТП. Рассмотрим трехуровневую систему, включающую уровень контроллеров, диспетчерский уровень и административный уровень.

АСУТП уровня контроллеров создается на основе Микро-монитора реального времени (Микро-МРВ). Эта программа размещается в РС-контроллере и осуществляет сбор данных с объекта, программно-логическое управление технологическими процессами и регулирование параметров по различным законам, а также ведение локальных архивов. Программа ведет постоянный контроль работоспособности УСО, сетевых линий, и в случае их выхода из строя автоматически переходит на резервные средства. При помощи Микро-МРВ можно создавать дублированные или троированные системы с горячим резервом.

Основу диспетчерского уровня управления составляют Мониторы реального времени (МРВ). МРВ ТРЕЙС МОУД – это сервер реального времени, осуществляющий прием данных с контроллеров, управление технологическим процессом, перераспределение данных по локальной сети, визуализацию информации, расчет ТЭП и статистических функций, ведение архивов.

На административном уровне АСУТП используются модули Supervisor. Supervisor предоставляет руководителю информацию о ходе и ретроспективе технологического процесса, статистических и технико-экономических параметрах предприятия. Эта информация может обновляться в режиме, близком к реальному времени (задержка 10 – 30 с). Кроме того, Supervisor дает возможность просматривать ретроспективу (осуществлять «плейбек») процесса как фильм на видеомагнитофоне. Графический «плейбек» архива дает в руки руководителя инструмент контроля работы диспетчерского комплекса и всего предприятия в целом.

В ТРЕЙС МОУД можно воспользоваться одной из следующих технологий разработки систем управления: проектирование системы в виде единого комплекса, сквозное программирование всех ее уровней, автоматическое построение и независимая разработка компонентов. Основные архитектурные и функциональные особенности инструментальной

системы можно охарактеризовать следующим образом. Она имеет интегрированную среду разработки, обеспечивает реальный многопользовательский режим, позволяет выбирать подход к проектированию системы управления, настраивать технологию разработки, программировать механизмы автоматического проектирования и обладает рядом других особенностей.

Системы, разработанные на базе ТРЕЙС МОУД, работают в энергетике, металлургии, нефтяной, газовой, химической и других отраслях промышленности и в коммунальном хозяйстве России. По числу внедрений в России ТРЕЙС МОУД значительно опережает зарубежные пакеты подобного класса.

К тенденциям развития распределенных систем управления можно отнести системный подход к построению системы с учетом стандартизации и унификации оборудования и программного обеспечения, единого подхода к многоуровневой структуре систем управления.

В заключение необходимо подчеркнуть, что проектирование или модернизация современных распределенных систем управления должны выполняться с учетом максимального удовлетворения требований на разработку системы управления, в том числе функциональных, технических, экологических, требований надежности; использования программно-аппаратных средств, отвечающих всем международным стандартам, гарантирующим соблюдение принципа открытости систем; экономической эффективности системы управления с учетом всего жизненного цикла системы и др. требований, определяемых спецификой и характером технологического процесса.

Литература

1. Андреев Е.Б. и др. Программные средства систем управления технологическими процессами в нефтяной и газовой промышленности / Андреев Е.Б., Попадько В.Е. – М.: Издательство РГУ нефти и газа им. И.М. Губкина, 2005. – 266 с.
2. Подьяпольский С.В., Родионов А.В., Соркин Л.Р. Распределенная система управления нового поколения Experion PKS компании Honeywell // Промышленные АСУ и контроллеры. – 2005. – №9. 1-6 с.
3. Синенко О.В. Анализ производственных процессов и подход к созданию комплексных систем управления производством / Синенко О.В. // Нефтяное хозяйство. – 2002. – №210. – С. 25-29.
4. Якобовский М.В. Распределенные системы и сети. М.: МГТУ "Станкин", 2000 г. – 118 с.
5. TDC 3000 System. System Technical Data: Honeywell Inc. 1990 – 22 p.
6. TRACE MODE, http://www.adastra.ru/

УДК 519.688

РАЗРАБОТКА ИНТЕЛЛЕКТУАЛЬНОЙ СИСТЕМЫ ПОИСКА АНАЛОГОВ РЕШЕНИЙ В ЭЛЕКТРОННОМ АРХИВЕ ПРОЕКТНОЙ ДОКУМЕНТАЦИИ С ПРИМЕНЕНИЕМ ИММУННОГО АЛГОРИТМА МУЛЬТИМОДАЛЬНОГО ПОИСКА

Мельникова Ирина Владимировна, доцент, Старооскольский технологический институт, Национальный исследовательский технологический университет «МИСиС», Россия, Старый Оскол, i-melnikova@mail.ru

Введение

Электронные архивы являются современным решением для повышения эффективности работы с информационными ресурсами [1]. Имеется большое количество как коммерческих, так и индивидуальных разработок систем работы с документацией и систем поиска документальной информации [2]. Однако, все предлагаемые, в том числе в научных публикациях, подходы базируются на гипертекстовом поиске. Несколько особняком стоят в этой области архивы проектно-конструкторской документации. В них хранятся разнородные по формам представления документы, однако подавляющее большинство составляет

документация в формате Autocad, что делает все виды гипертекстового поиска малоэффективными. Кроме того, очень часто возникает необходимость поиска похожих проектов, поскольку часто новые проекты представляют собой комбинацию ранее разработанных типовых решений.

Задача обработки накопленной в ходе деятельности предприятия документации и извлечения из нее *знаний* является классической задачей Data Mining. К числу методов исследования данных в Data Mining можно также отнести бурно развивающийся в последнее время аппарат иммунных сетей.

Иммунные сети являются естественным продолжением практики применения механизмов биологических систем для решения множества задач в широкой области приложений. Перспективы применения имунных алгоритмов в системах Data Mining рассматривались еще в базовых работах L. N. De Castro и Тиммиса [3]. Процессы, происходящие при обработке информации естественными системами и принципы их функционирования, поражают своей эффективностью, экономичностью и быстродействием. Разработка высокоэффективных ИИС находится на начальном этапе, поэтому они недостаточно распространены, но имеют огромный потенциал развития [4].

1. Постановка задачи исследования

Постановка задачи исследования формулируется следующим образом: необходимо разработать интеллектуальную информационно-поисковую систему, решающую задачу поиска аналогичных проектов в архиве проектной организаци на основе биологического подхода ИИС.

Для классификации проектов предлагается поставить в соответствие каждому проекту бинарную последовательность, в которой каждой позиции соответствует какое-либо проектное решение, согласно разработанному для данной тематики перечню видов проектов и входящих в их состав отдельных решений. Тогда новому проекту будет соответствовать некоторая известная бинарная последовательность, где наличию или отсутствию какого-либо решения будут соответствовать 0 или 1. При этом задача сводится к поиску в полученном после сериализации файле сигнатур проектов, наиболее соответствующих заданной. Так как идентичные проектные решения могут встречаться в разных проектах, то результат поиска предполагает несколько вариантов наиболее схожих проектов.

Традиционными методами подобные задачи решаются плохо. В связи с этим актуально применение биологического подхода Искусственных Иммунных Систем (ИИС), построенных на принципах обработки информации молекулами белков [5].

Для решения данной задачи наиболее подходящим представляется иммунный алгоритм мультимодального параллельного поиска [6]. Существует несколько разновидностей алгоритма мультимодальной оптимизации – основные это aiNet и модифицированный алгоритм CLONALG.

2. Разработка теоретических основ и алгоритма построения ИПС

Естественная иммунная система имеет уникальную способность вырабатывать новые типы антител и отбирать наиболее подходящие из них для взаимодействия с попавшими в организм антигенами. Методом проб и ошибок иммунная система вырабатывает огромное количество антител против бесчисленного множества неизвестных антигенов. Важной характеристикой процесса адаптации системы иммунитета при ее взаимодействии с поступающими из внешней среды вирусами и бактериями является обеспечение разнообразия типов антител. С математической точки зрения поддержание разнообразия в иммунной системе можно трактовать как задачу оптимизации мультимодальной функции, имеющую кратное (не единственное) решение.

Предлагаемый алгоритм обеспечивает возможность одновременного хранения нескольких векторов поиска для нахождения кратных решений. Для этого вводится индекс

разнообразия, а вектор приближений сохраняется, аналогично механизму поддержания клеток памяти в иммунной системе.

Будем считать, что имунная система является аналогом имунного алгоритма.

Рассмотрим имеющийся набор сигнатур проектов как антитела, а бинарную последовательность нового проекта – как антиген.

Антиген x представляет собой совокупность проектных решений, сформированных пользователем.

Аффиности ax_v и $ay_{v,w}$– это степень схожести как имеющихся проектов в базе между собой, так и сформированного пользователем с ними.

B–лимфоциты c_v– массив значений, участвующий в процессе удаления из рассмотрения неподходящих проектных решений.

Ожидаемый масштаб выработки e_v – значение, указывающее, какие проекты не удовлетворяют результатам запроса, и собственно удаляются, а какие остаются.

Тогда алгоритм поиска можно представить в следующем виде:

1. *Распознавание антигена*, т.е. получение бинарной последовательности, для которой необходимо найти аналоги. Данный шаг соответствует определению вида задачи оптимизации, а бинарная последовательность нового проекта (антиген) представляет собой критерий, по которому будет осуществляться оптимизация.
2. *Выработка антител*, т.е. извлечение из памяти имеющихся бинарных последовательностей–антигенов. Этот шаг соответствует «вспоминанию» полученного в прошлом успешного решения, а для нашей задачи– выполненных ранее проектов.
3. *Вычисление аффинитета*, т.е. определение набора B–лимфоцитов, индуцирующих наиболее подходящие антитела. Для этого вычисляются функции аффинитета: аффинность антител для каждой пары антител

$$ay_{v,w} = \frac{\sum_{i=1}^{M} s_i}{\sum_{j=1}^{M}\sum_{i=1}^{S} p_{ij}} \qquad (1)$$

где M – количество бит в бинарной последовательности, S – количество готовых проектов-антител, s_i –вероятность равенства i–того бита (отдельной работы выполняемой в проекте) текущего антигена единице, т.е ; p_{ij} – вероятность равенства i–того бита текущего антигена i–тому биту j–того антигена; и аффинность антитела к антигену

$$ax_v = opt_v, \qquad (2)$$

где величина opt_v характеризует силу связи между возможным решением-антителом и искомым проектом – антигеном и вычисляется посредством логического умножения соответствующих бинарных последовательностей с последующим подсчетом количества ненулевых бит результата, т.е. заданными к поиску видами отдельных проектных решений. Кроме того, сохраняются номера единичных бит антигена, соответствующих единичным битам антитела, т.е. найденными аналогами искомых работ входящих в состав проекта. Собственно, на этом шаге происходит поиск оптимального решения

4. *Дифференцировка лимфоцитов*, т.е. сохранение подходящего решения для следующего шага поиска. На данном шаге некоторые B–лимфоциты становятся клетками памяти, а также супрессорными клетками. Супрессорные клетки необходимы для удаления избытка кандидатов на решение. Выбор таких B–лимфоцитов происходит по следующей формуле:

$$c_v = \frac{1}{N}\sum_{w=1}^{N} ac_{v,w} > T_c, \text{ , где} \qquad ac_{v,w} = \begin{cases} 1 & \text{если } ay_{v,w} \geq T_{ac1}; \\ 0 & \text{иначе } ay_{v,w} < T_{ac1}, \end{cases} \tag{3}$$

T_c – пороговое значение для В–лимфоцита; T_{ac1} – пороговое значение для антигена.

5. *Размножение и подавление антител*. Ожидаемый масштаб e_v выработки антител задается формулой

$$e_v = \frac{ax_v \prod_{s=1}^{S}\left(1 - as_{v,s}\right)}{c_v \sum_{i=1}^{N} ax_i}, \qquad \text{где} \quad as_{v,s} = \begin{cases} ay_{v,s} & \text{если } ay_{v,s} \geq T_{ac2}; \\ 0 & \text{иначе } ay_{v,s} < T_{ac2}. \end{cases} \tag{4}$$

Соотношение (4) регулирует концентрацию и разнообразие проектов-аналогов (антител) в популяции лимфоцитов. Если антитело приобретает более высокий аффинитет к искомому проекту-антигену, то оно размножается, но при этом размножение антител, концентрация которых слишком велика, подавляется. Наряду с отслеживанием локальных максимумов это соответствует поддержанию разнообразия направлений поиска. Данный механизм поиска полностью соответствует решаемой задаче, т.к. отдельные проектные решения могут содержаться в локальных максимумах, а также в слабо релевантных решениях. Это определяется спецификой проектирования, так как однозначно отнести проект к той или иной категории часто невозможно.

6. *Размножение антител*. Для ответа на ранее не встречавшиеся антигены, т.е. новый проект, происходит образование новых лимфоцитов вместо антител (проектов найденных ранее), удаляемых на предыдущем шаге. В рамках иммунного алгоритма это позволяет генерировать разнообразие антител на основе генетических операторов репродукции, например, под действием операторов мутаций или скрещивания. Необходимым условием этого является более высокая эффективность таких операторов по сравнению с генерацией антител на чисто случайной основе.

Данный алгоритм строится на понятии формального белка (ФБ) [7], в котором закодированы бинарные последовательности уже выполненных и содержащихся в каталоге организации проектов, одна последовательность – один проект в каждом белке. Новый проект при поступлении на разработку кодируется аналогичнвм образом, впоследствии, при занесении в категорию выполненных, его бинарная последовательность может быть откорректирована по факту выполненных работ. Описание формального белка проистекает из попытки описания его геометрической структуры посредством аппарата кватернионов [8].

Формальный белок представляет собой упорядоченную пятерку, образованную следующими компонентами: $P=\{n,U,Q,V,v\}$,

1. Количеством связей n>0, равным количеству бит бинарной последовательности описываемого проекта;
2. Множеством углов $U = \{\varphi_k, \psi_k\}, k = 1, \dots, n,$ где $-\pi \leq \varphi_k \leq \pi$, $-\pi \leq \psi_k \leq \pi$. В нашем случае углы вращения представляют собой биты бинарной последовательности;
3. Множеством единичных кватернионов $Q=\{Q_0,Q_k\}$, где кватернионы $Q_k = Q_k(\varphi_k, \psi_k)$ определяются формулами (5) и (6), и результирующий кватернион формального белка Q_0 определяется как их произведение:
$$Q_0=Q_1Q_2\dots Q_n;$$
4. Множеством коэффициентов $V=\{v_{ij}\}$, i=1,2,3,4, j≥i;
5. Функцией v (без индекса), определенной над элементами результирующего кватерниона Q_0 посредством следующей квадратичной формы:
$$v = -\sum_{j>i} v_{ij} q_i q_j \tag{7}$$

Антитело, т.е. задание на проектирование, представляется в виде формального белка. В–лимфоцит представляет собой более сложную структуру, способную генерировать формальные белки–антитела с тем, чтобы они связывались с антигенами, т.е. с имеющимися в нашем распоряжении проектами.

Формальный B-лимфоцит – упорядоченная четверка $B - cell \langle P, Ip, Is, Im \rangle$, где

P – рецептор лимфоцита, представляющий собой ФБ; Ip – индикатор состояния рецептора; Is – индикатор состояния лимфоцита; Im – индикатор мутации.

Поведение B-лимфоцита определяется следующими правилами:

1. B-лимфоцит может находиться только в следующих состояниях: *Is = {0,1,2}*

Is=0 обозначает смерть, когда *B*-лимфоцит уничтожен;

*Is=1*обозначает распознание, когда рецептор *p B*-лимфоцита может связываться с другим ФБ;

Is=2 обозначает размножение лимфоцита, когда *B*-лимфоцит делится на две одинаковые копии, состояние которых устанавливается в *Is=1*, а состояние их рецепторов определяется индикатором мутации *Im*;

Переход из состояния *Is=1* в состояние *Is=2* может происходить только в результате связывания рецептора *P* с другим ФБ.

2. Индикатор мутации *Im={0,1}* родительского лимфоцита определяет состояние рецепторов его копий следующим образом:

Im=0 рецепторы наследуются от родительского B-лимфоцита (без мутации);

Im=1 рецепторы изменяются (с мутацией).

Другими словами, главным свойством формального B-лимфоцита является его размножение (или смерть) в результате свободного связывания (или не связывания) с некоторым ФБ.

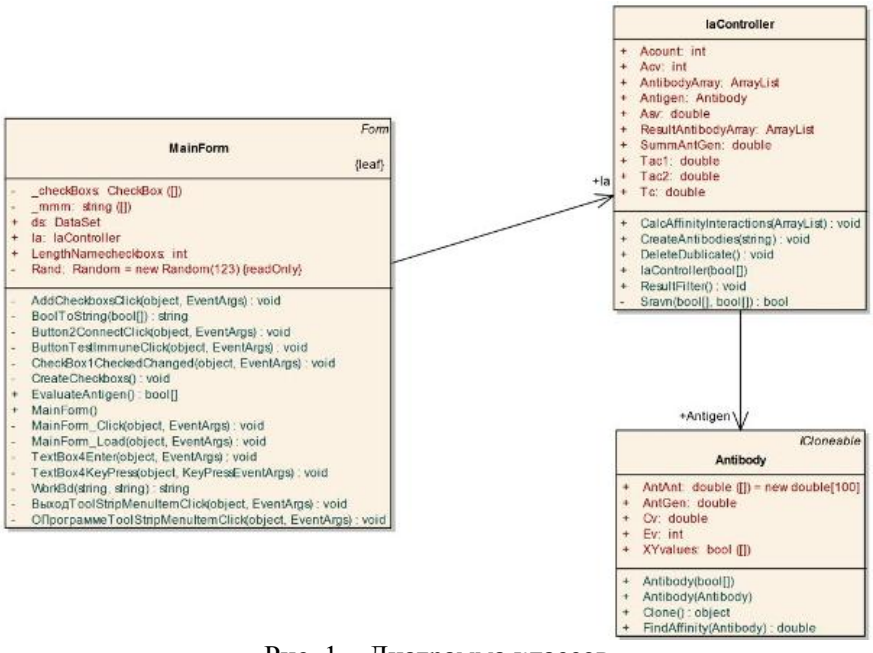


Рис. 1 – Диаграмма классов

Реализация данного алгоритма выполнялась с использованием объектно-ориентированного подхода (ООП). Одним из наиболее важных понятий ООП является класс. Класс представляет собой дальнейшее развитие концепции типа и объединяет в себе задание не только структуры и размера переменных, но и выполняемых над ними операций. Разработка производилась в среде программирования Microsoft Visual C# 2008 Express Edition Функциональная схема поисковой системы реализованной на базе предложенного алгоритма представлена на рис.1 и состоит из 3 классов:

1) *Класс Mainform* обеспечивает связь с пользователем, подключение к базе данных и работу остальных модулей.

2) *Класс Antibody* описывает свойства антитела, а также методFindAfinity для нахождения аффинности для пары антител.

Модуль FindAfinity реализует алгоритм для нахождения аффинности как между парами анитител так и антитела с антигеном.

3) КлассI aController реализует алгоритм параллельного поиска и вспомогательные методы.

Модуль Calcaffinity Iterations реализует алгоритм параллельного поиска.

Модуль Delete Dublicate предназначен для отсеивания одинаковых антител, которые образовались на шаге клонирования метода.

Модуль Result Filter объединяет части результатов в один результирующий массив, блок-схема *модуля* приведена на рис 2.

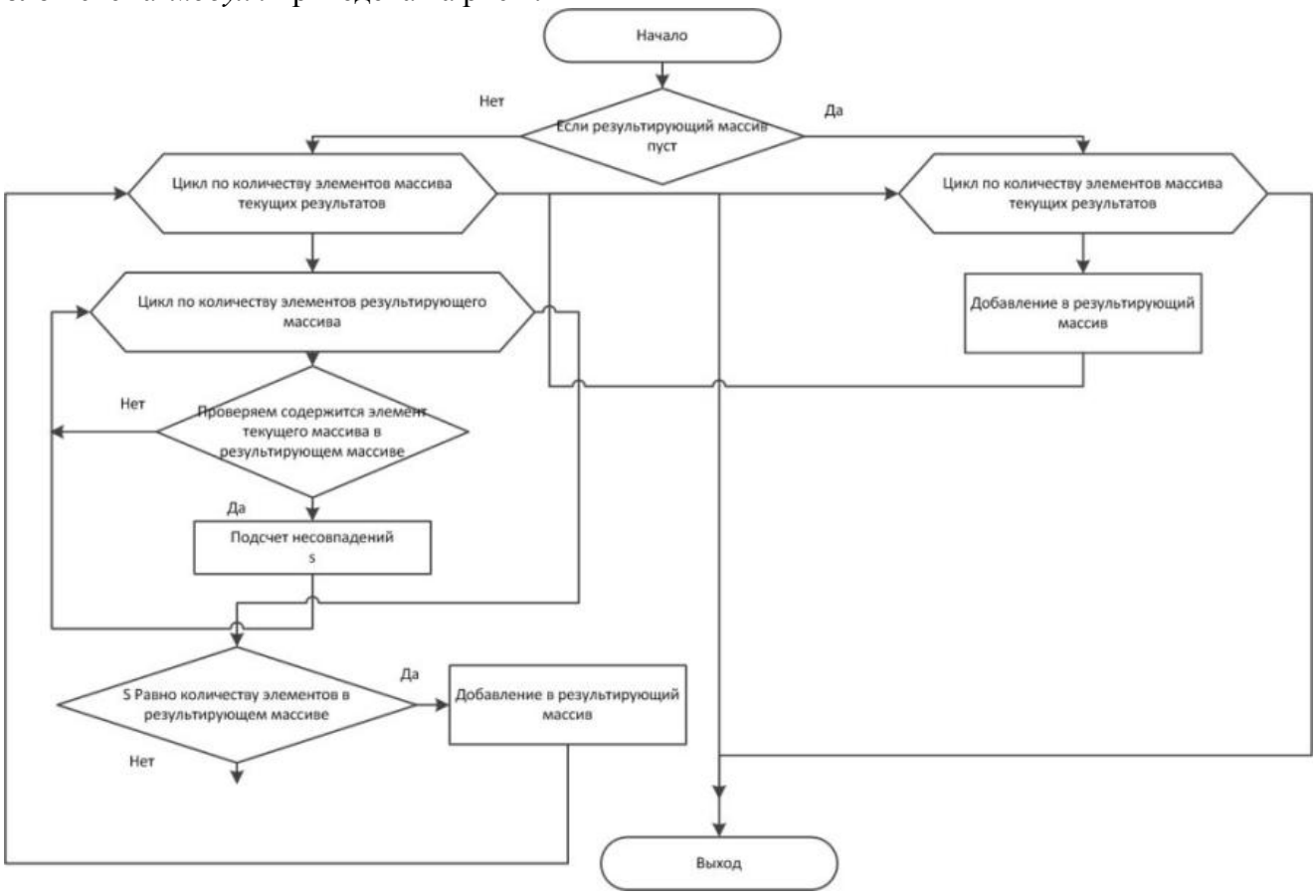


Рис. 2 – Блок-схема модуля Result Filter

Было проведено тестирование на каталоге электронного архива проектной организации, специализирующейся в области электротехнических решений, содержащем около 1000 проектов. Оценка эффективности поисковой системы производилась по следующей методике [9].

Степень релевантности пары *(d,s)*: $\eta(d,s) = \frac{1}{m}\sum_{j=1}^{m}\eta_j(d,s)$ (8)

Коэффициент полноты: Коэффициент точнности:

$$k_n(s) = \frac{\sum_{d\in M} r(d,s)\varphi(d,s)}{\sum_{d\in M} r(d,s)} \qquad k_n(s) = \frac{\sum_{d\in M} r(d,s)\varphi(d,s)}{\sum_{d\in M} r(d,s)} \qquad (9), (10)$$

d – проект из множества проектов; s –запрос проекта; (d,s) – релевантная пара4

r(d,s) – отношение релевантности; M – множество проектов;

r(d,s) = 1, если проект d релевантен запросу s

0, если проект d не релевантен запросу s (11)

φ(d,s) – отношение выдачи, где

φ(d,s) = 1, если ИС выдает проект d на запрос s

0, если ИС не выдает проект d на запрос s (12)

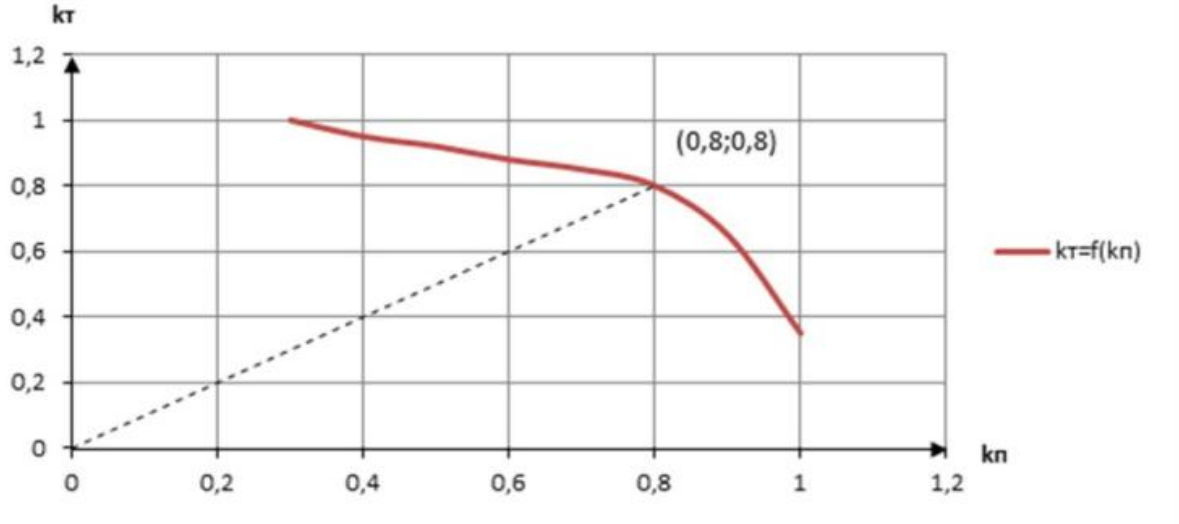


Рис. 3 – Кривая оптимальных характеристик качества поиска

Выводы

В данной работе обосновано применение искусственных иммунных сетей к задаче мультимодальной оптимизации для решения задачи поиска в электронном архиве проектной документации. Предлагается оригинальная система классификации и кодирования проектов, позволяющая полностью описать все проектные решения, входящие в состав проекта, сведя их в конечном итоге к файлу сигнатур проектов.

Литература

1. И.В. Мельникова «Анализ существующих проблем хранения проектной документации в архивах предприятий и организаций. Международная научно-практическая конференция «Образование, наука, производство», г.Ст.Оскол 20-21 ноября 2008г.
2. И.В. Мельникова «Обзор некоторых подходов к организации электронных архивов проектных организаций. Международная научно-практическая конференция «Образование, наука, производство и управление», г.Ст.Оскол 20-21 ноября 2008г.
3. de Castro L.N., Timmis J. Artificial Immune Systems. A New Computational Intelligence Paradigm. – N.-Y.: Springer, 2002.
4. Julie Greensmith New Frontiers For An Artificial Immune System Digital Media Systems Laboratory, HP Laboratories Bristol, HPL-2003-204, October 7th , 2003*.
5. Tarakanov A.O. Formal peptide as a basic of agent of immune networks: from natural prototype to mathematical theory and applications. Proceeding of the I Int. workshop of central and Eastern Europe on Multi. – Agent Systems, 1999.
6. L. de Castro and J. Timmis. An artificial immune network for multimodal function optimization. In Proc. of the Congress on Evolutionary Computation (CEC), volume 1, pages 699–704, Los Alamitos, CA, USA, 2002. IEEE Computer Society.
7. Goldberg D. Genetic algorithms in search, optimization, and machine learning. N.Y.: Addison-Wesley, 1989.
8. Cantor C., Schimmel P. Biophysical Chemistry. Part I. The conformation of biological macromolecules. San Francisco: W.H. Freeman & Co. 1980
9. Соколов А.В.«Методика оценки максимально возможных значений показателей эффективности поиска текстовой информации», Информационные технологии, – 2009. №5, стр. 18-24

УДК 004.4’41:378

ЭЛЕКТРОННОЕ УЧЕБНОЕ ПОСОБИЕ «ДЕТЕРМИНИРОВАННЫЙ КОНЕЧНЫЙ АВТОМАТ»

Кузьмина Тамара Михайловна, к. ф-м. н., доцент, Московсковский государственный текстильный университет им. А.Н.Косыгина, Россия, Москва, kuzmina_t_m@mail.ru

Конечные автоматы широко используются в программировании. Они используются при написании трансляторов [1], в частности, для проведения лексического анализа; при программировании мобильных устройств [2] и т.д. Студенты специальности САПР изучают конечные автоматы в курсе «Лингвистическое и программное обеспечение САПР». Конечный автомат определяется как некоторая абстрактная машина, которая позволяет по ленте, содержащей строку, выдать ответ: допустима строка или нет. Термин «автоматный язык» закреплен за языками, распознаваемыми именно конечными автоматами, а не какими-либо более широкими семействами автоматов (например, автоматами с магазинной памятью или линейно ограниченными автоматами).

Для ускорения процесса обучения был написана программа, являющаяся эмулятором детерминированного конечного автомата (ДКА). При запуске программы студент при желании может вызвать справку и прочитать определение ДКА, может сразу приступить к практической работе, в которой ему надо определить автомат, решающий конкретную задачу. Для этого сначала определяется входной алфавит, затем таблица переходов, начальное и заключительные состояния.

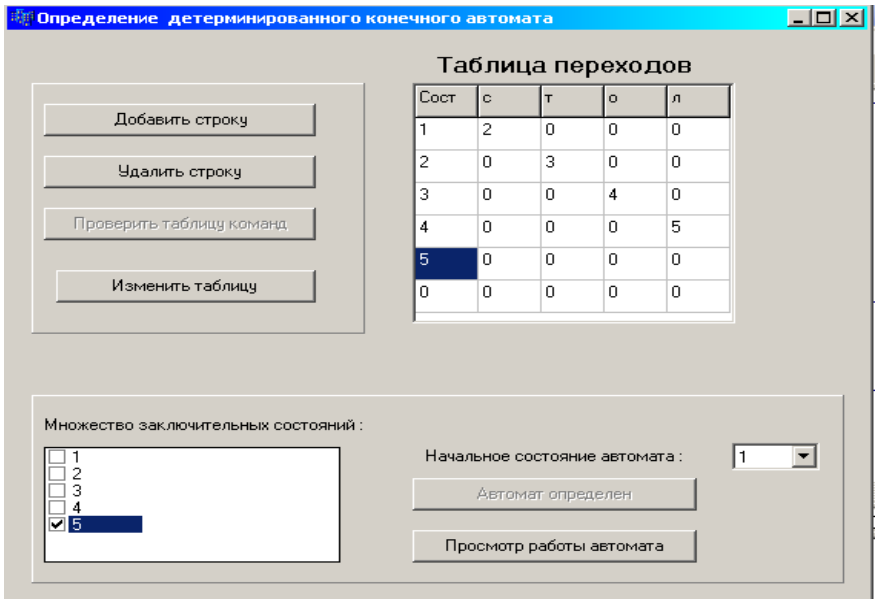


Рис. 1 – Основное окно программы

На рис. 1 показано окно, с которым работает студент, определяя таблицу переходов, начальное и заключительные состояния. В данном случае строится автомат, принимающий язык, содержащий одно слово – «стол». Изначально таблица переходов пуста, но заголовки столбцов уже определены заданным алфавитом. Студент, задавая входной алфавит, может просто вводить слова языка, не заботясь о том, что буквы будут повторяться. Программа удалит ненужные повторения, а затем сформирует заголовки столбцов таблицы переходов.

Студент заполнит таблицу, а затем программа проверит ее на корректность. Поскольку в первом столбце должны быть перечислены все состояния автомата, то программа проверяет, чтобы в остальных столбцах при задании переходов использовались именно эти значения. Программа позволяет задать язык, содержащий любое количество слов и, соответственно, автомат, содержащий большое количество состояний. Но, поскольку студенты работают с языками, содержащими по 3-4 слова, то в программе не предусмотрены

средства, упрощающие работу с языками большой мощности (такими, как распахивание окна на весь экран и пр.).

После задания таблицы переходов, становится доступной нижняя панель, на которой студент выбирает начальное и заключительные состояния автомата. На этом определение автомата заканчивается, и можно просмотреть его работу. В нашем примере состояния автомата обозначаются цифрами от 0 до 4, т.е. в порядке возрастания, но это не обязательно, т.к. номера состояний могут быть любыми и располагаться в любом порядке. Эти номера будут перенесены программой в списки, предназначенные для выбора начального и заключительного состояний. При построении таблицы переходов довольно часто встречается ситуация, когда автомат попадает в состояние, заведомо не переходящее ни в одно из заключительных состояний, номер этого состояния можно выбрать равным 0. Это не обязательно, но удобно, поскольку программа пустые клетки заполняет нулями.

Автоматическое формирование заголовков таблицы переходов, списков состояний, из которых выбираются начальное и заключительных состояния, не только упрощает работу с программой, но и демонстрирует правила построения конечных автоматов.

На Рис. 2 показано окно, в котором демонстрируется работа автомата. Обычно ДКА представляется как устройство, имеющее неподвижную ленту, на которой написано анализируемое слово и управляющее устройство, перемещающееся по ленте слева направо. Управляющее устройство имеет считывающую головку, которая обозревает один символ.

В предлагаемой программе эта конструкция изменена: устройство сделано неподвижным, а лента перемещается справа налево. На Рис.2 это перемещение показано стрелкой. Обозреваемое слово разбивается на две части, слева от считывающей головки находится проанализированная часть, справа – непроанализированная часть слова. Тот символ, который анализируется, в данный конкретный момент выделяется другим цветом, этот символ на следующем шаге будет перемещен в левую часть ленты.

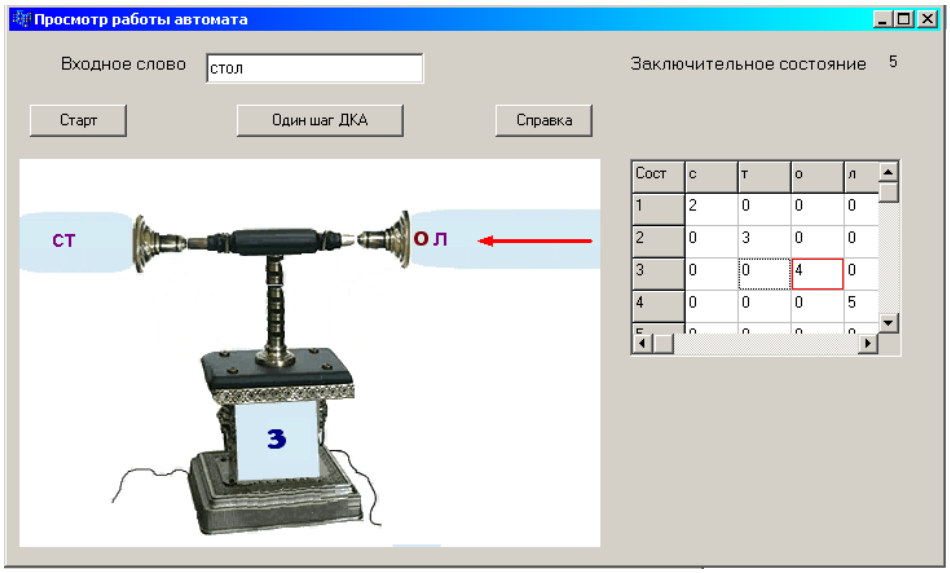


Рис. 2 – Демонстрация работы ДКА

Пользователь вводит обрабатываемое слово в специальное поле ввода. При нажатии на кнопку «Старт» слово переносится на ленту автомата справа от считывающей головки, автомат переходит в начальное состояние, которое пишется на самом управляющем устройстве. При нажатии на кнопку «Один шаг работы» выполняется следующий шаг работы ДКА. В зависимости от того, какой символ обозревается (этот символ стоит справа от считывающей головки и выделен другим цветом), и от того, в каком состоянии находится автомат (номер состояния, в котором находится ДКА, пишется на самом управляющем устройстве), автомат переходит в новое состояние, возможно, совпадающее со старым состоянием, и начинает обозревать следующий символ. Новое состояние автомата определяется таблицей переходов. В таблице переходов на каждом шаге тоже происходят

изменения – смещается красный квадратик. Красный квадратик всегда располагается на пересечении столбца, соответствующего обозреваемому символу, и строки, соответствующей номеру текущего состояния автомата, он выделяет номер состояния автомата, в который перейдет автомат на следующем шаге.

На рис. 2 автомат находится в состоянии 3 и обозревает символ «о», в таблице переходов выделена красным квадратиком ячейка, стоящая на пересечении столбца под именем «о» и строки, соответствующей состоянию 3. На следующем шаге автомат перейдет в состояние 4 и будет обозревать символ «л».

По завершению работы автомата – просмотра всех символов слова – автомат выдает сообщение о том, принимается слово или нет. Это сообщение можно проверить, выяснив совпадает ли состояние автомата, в котором он остановился, с одним из заключительных состояний или нет.

Данная программа была предложена студентам в качестве необязательного вспомогательного пособия при выполнении лабораторной работы «Использование ДКА при распознавании слов определенного языка», в которой надо было построить ДКА, а затем написать программу на языке C#, моделирующую работу построенного автомата. Почти 80% студентов использовали данное электронное пособие для разработки и проверки «бумажной версии» своего ДКА, а затем приступали к написанию программы. Многие из оставшихся 20% студентов, не затруднившихся написать ДКА сразу на листке бумаги, при обнаружении неточностей в работе программы, обращались к предлагаемой программе позднее, для того, чтобы еще раз проверить свою «бумажную» версию ДКА.

После введения в учебный процесс данного пособия среднее время, затрачиваемое студентами на выполнение лабораторной работы, значительно уменьшилось, а преподаватель был освобожден от работы по проверке «бумажных версий» ДКА.

Программа также может быть использована при самостоятельном изучениии ДКА, при заочной форме обучения или дистанционном образовании.

Литература

1. Кузин Л.Т. Основы кибернетики, т. 2. – М.: Энергия, 1979 г.
2. Салмре И. Программирование мобильных устройств на платформе .Net Compact Framework. – М.: Вильямс, 2006.

УДК 004.78

РЕАЛИЗАЦИЯ МОДУЛЯ РЕЦЕНЗИРОВАНИЯ В ИНФОРМАЦИОННОЙ СИСТЕМЕ ПРОВЕДЕНИЯ НАУЧНЫХ КОНФЕРЕНЦИЙ

Олейник Павел Петрович, к.т.н., Системный архитектор программного обеспечения, ОАО «Астон», Россия, Ростов-на-Дону, xsl@list.ru
Игумнов Евгений Александрович, Шахтинский институт (филиал) Южно-Российского государственного технического университета (Новочеркасского политехнического института), Россия, Шахты, gyma@bk.ru
Свечкарёв Евгений Андреевич, Шахтинский институт (филиал) Южно-Российского государственного технического университета (Новочеркасского политехнического института), Россия, Шахты

Основным способом ознакомления научного сообщества с результатами проведённых исследований в настоящее время является опубликование доклада на научной конференции. Каждый современный ВУЗ ежегодно организует конференции различного уровня (Внутривузовская, Межвузовсткая, Всероссийская, Международная). При проведении подобного мероприятия основной задачей является каталогизирование докладов, рецензирование и своевременное информирование авторов о судьбе их работы. В данной статье рассматривается реализация модуля рецензирования информационной системы, которая разработана для каталогизирования научных работ, присланных авторами на конференцию «Объектные системы» [1-3].

При разработке новой или доработке существующей информационной системы необходимо сформулировать ряд критериев оптимальности (КО), определяющих функциональные требования к возможностям, присутствующим в готовом программном приложении. При проектировании модуля рецензирования были выделены следующие критерии оптимальности:

1. Возможность отслеживания истории рецензирования каждой отдельной статьи. Рецензирование – одна из наиболее важных и ответственных задач, которые ложатся на членов программного и организационного комитетов. Основная проблема здесь в том, что сама процедура выполняется в относительно короткий временной интервал и при этом одну и ту же статью могут рецензировать несколько рецензентов.
2. Реализовать возможность контроля изменений в файлах, исправленных авторами (или рецензентами) после соответствующей рецензии. Авторам рецензируемых докладов может потребоваться внести изменения в присланную статью в соответствии с указанными замечаниями.
3. Предусмотреть возможность рекомендации статей (после соответствующей доработки) в журналы, которые являются информационными партнёрами конференции. Любая конференция не может проводиться изолированно и, как правило, имеет обширные связи с информационными партнёрами, которыми являются журналы. На основании коллегиального голосования некоторые статьи, присланные на конференцию, могут быть рекомендованы к опубликованию в этих журналах.
4. Реализовать возможность объявления номинаций в проводимой конференции и выявить лауреатов. Определение номинаций, наград и победителей – одна из ключевых задач проведения конференций, так как подобные процедуры способствуют активизации научной деятельности авторов, особенно молодых учёных.

Структура классов проектируемого приложения, соответствующая выделенным критериям оптимальности, представлена на рис. 1. Данная иерархия классов построена путём доработки классов, присутствующих на соответствующих диаграммах, рассматриваемых в работе [2-3] и является логической моделью разработанной БД, которая проектировалась в понятиях объектно-ориентированной парадигмы.

Кратко рассмотрим назначение выделенных классов. Класс BaseConference является базовым абстрактным классом, предназначенным для определения иерархий конкретных классов (унаследованных), определяющих типы научных конференций. Производный класс OwnConference представляет конференцию, проводимую собственным ВУЗом. Информационные партнёры, поддерживающие конференцию (как правило, журналы, включённые в список ВАК), в которых будут опубликованы доклады лучших авторов, представляются экземплярами класса SupportEdition. Для сохранения номинаций выделен класс Nomination. При определении номинаций, в которых может победить статья, имеется возможность указать бонус победителю (например, бесплатное участие в данной или в следующей конференции).

Статьи, присланные на конференцию, представляются экземплярами класса OwnConferenceArticle. При этом каждый автор физически присылает по электронной почте несколько файлов на конференцию (файл с текстом доклада, графические файлы с изображениями рисунков, заявки и т.п.), для сохранения которых используются классы, производные от абстрактного File.

Каждый учёный может быть включён в качестве члена программного либо организационного комитетов несколько раз, поэтому было выделено два класса Scientist и Member соответственно. Рецензия определённого учёного на конкретную статью представляется экземпляром класса Review, основным атрибутом которого является ReviewResult, позволяющий отслеживать результат рецензии (принята ли статья на конференцию «как есть», статье требуется доработка/переработка, статья отклоняется). Информация о рекомендациях статьи в журналы, включённые в список ВАК, сохраняется в экземплярах класса RecommendationInSupportEdition.

Рассмотрим соответствие разработанной иерархии каждому из выделенных критериев оптимальности. Логическая модель соответствует $КО_1$, т.к. имеется возможность отслеживания истории рецензирования каждой отдельной статьи, что реализуется с помощью создания нескольких экземпляров Review для определённой статьи.

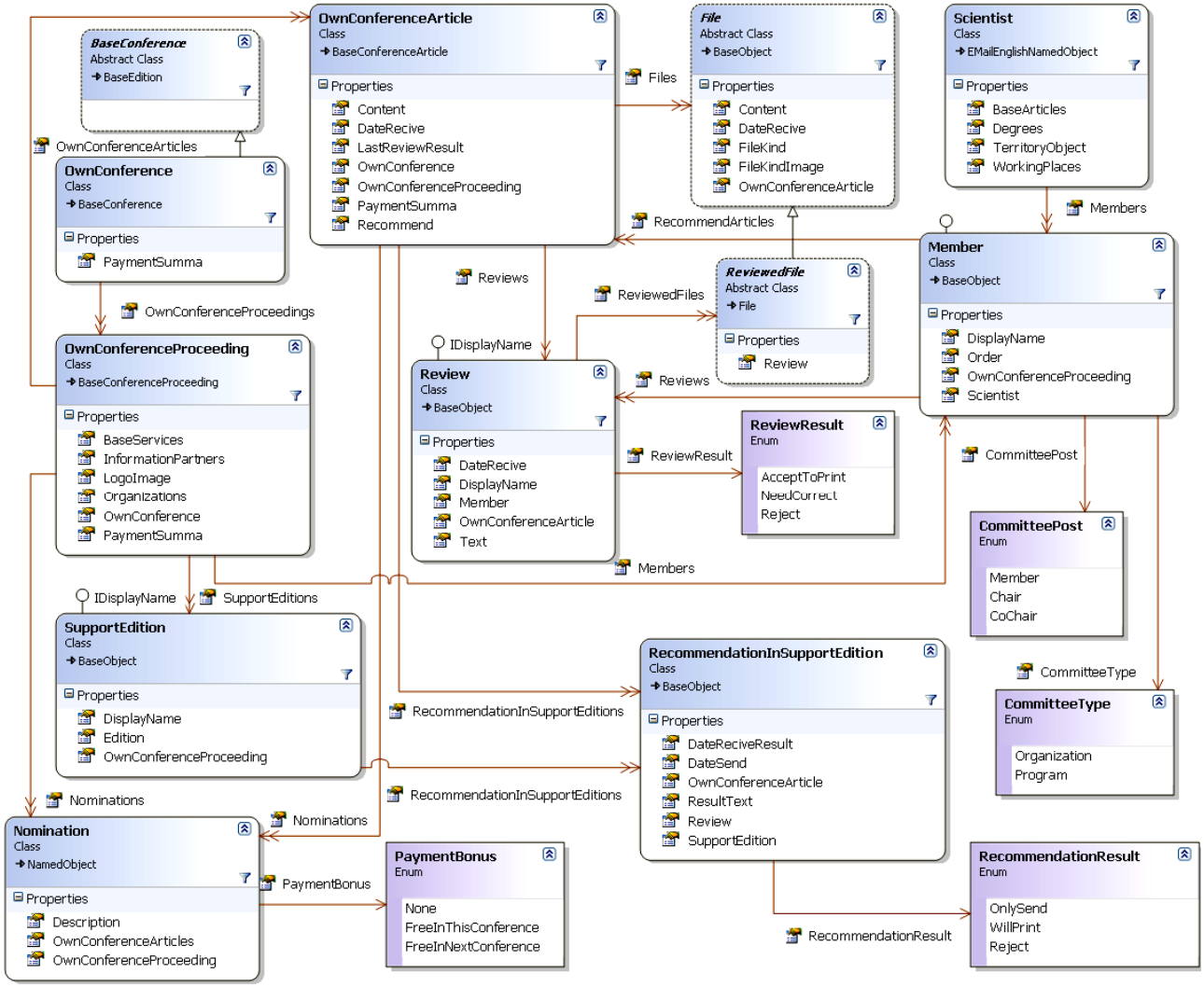


Рис. 1 – Диаграмма классов модуля рецензирования научных работ

Система также соответствует $КО_2$, т.к. реализована возможность контроля изменений в файлах после соответствующей рецензии в том случае, когда статья автора не удовлетворяет требованиям рецензентов. Это осуществлено с помощью указания в каждом присланном файле информации о том, после какой рецензии он был создан (откорректирован).

Спроектированный набор классов также удовлетворяет требованиям $КО_3$, потому что предусмотрена возможность рекомендации статей в журналы, которые являются информационными партнёрами конференции.

Также не нарушены требования $КО_3$, т.к. реализована возможность объявления номинаций в проводимой конференции и определения лауреатов, чьи доклады являются победителями.

Проанализировав описанное выше, можно сделать вывод, что разработанная иерархия классов полностью удовлетворяет всем выделенным критериям оптимальности. На рис. 2 изображён внешний вид графического пользовательского интерфейса, который представляет основные формы, отображающие экземпляры основных описанных ранее классов (см. рис. 1).

Из рис. 2 видно, что в верхней части отображается список присланных докладов, сгруппированный по названию проводимой конференции. В средней части диалоговой формы отображается подробная информация о конкретной работе с указанием файла со статьёй, который является окончательной версией доклада (с учётом всех исправлений). В нижней представлен список рецензий, полученных на статью с указанием текста рецензии и результата (принята статья или нет). В том случае если рецензент требует от автора внести корректировку (изменения) текста присланной работы, исправленные файлы прикрепляются и отображаются в той части формы, которая озаглавлена «Исправленные файлы». Закладка «Рекомендации в поддерживающие журналы/конференции» содержит данные в том случае, если, по мнению нескольких рецензентов, статья авторов может быть

опубликована (после соответствующей доработки) в журнале, выступившем в качестве информационного партнёра конференции. Номинации, в которых победил доклад, отображаются на закладке «Победитель номинаций».

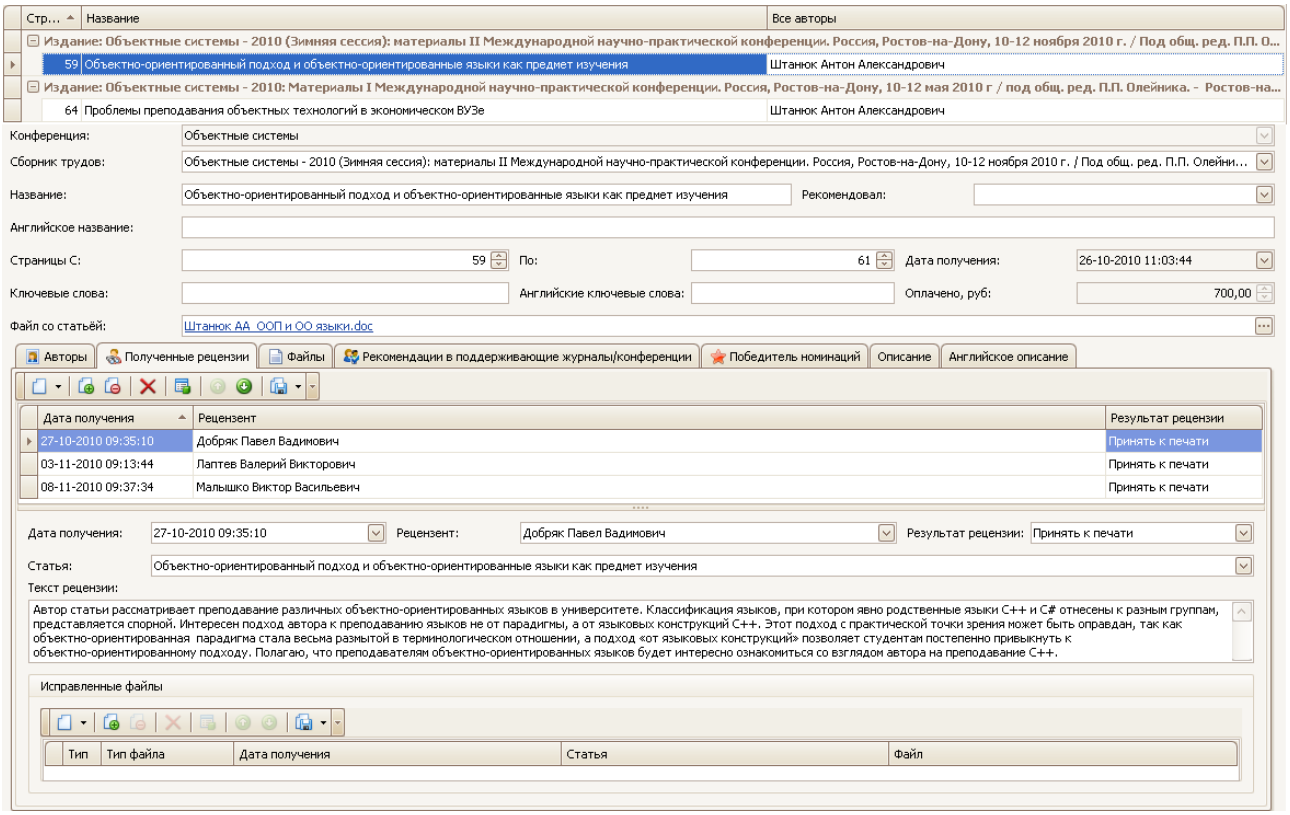


Рис. 2 – Графический интерфейс модуля рецензирования научных работ

Дальнейшим развитием системы является разработка модулей, позволяющих учитывать поступление денежных средств на конференцию (организационных взносов) и учёта затрат, связанных с выполнением типографских работ (публикация сборников, печать именных сертификатов, подтверждающих участие в конференции) и т.п.

Литература

1. Международная научно-практическая конференция «Объектные системы», www.objectsystems.ru
2. Олейник П.П., Игумнов Е.А., Свечкарёв Е.А. Критерии оптимальности информационной системы каталогизирования научных работ // Объектные системы – 2010: Материалы I Международной научно-практической конференции. Россия, Ростов-на-Дону, 10-12 мая 2010 г / под общ. ред. П.П. Олейника. – Ростов-на-Дону, 2010. С. 25-29.
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.

УДК 519.711

СИСТЕМА КОНТРОЛЯ ДИАГРАММНЫХ ЯЗЫКОВ

Афанасьев Александр Николаевич, к.т.н., профессор, Ульяновский государственный технический университет, Россия, Ульяновск, a.afanasev@ulstu.ru

Гайнуллин Ринат Фаязович, аспирант, Ульяновский государственный технический университет, Россия, Ульяновск, r.gainullin@gmail.com

Введение

В программной инженерии существует проблема успешности проектирования и создания сложных информационных систем, интенсивно использующих программное обеспечение. Степень успешности разработок таких систем, рассчитанная через число проектов, завершившихся в соответствии с исходными замыслами и планами, чрезвычайно низка (около 30%).

К числу основных причин, приводящих к неудачам, относятся: низкая степень взаимодействия разработчиков системы с заказчиками; недостаточная степень понимания и взаимопонимания в круге лиц, вовлеченных в индивидуальные и коллективные проектные работы; проблемы с адекватностью и полнотой требований к информационной системе.

Для улучшения ситуации с проблемами взаимодействия в инженерии программного обеспечения используют схемы и диаграммы. Они позволяют внести ясность в общение между заказчиком, проектировщиком и исполнителем информационной системы [1] .

В практике проектирования активно применяются диаграммные графические языки UML, IDEF, ARIS, SDL, ER, DFD. Однако инструментарии, реализующие технологии проектирования, не содержат эффективных средств контроля топологии диаграмм, что приводит к ошибкам, имеющим серьезные последствия.

В настоящей работе предлагается автоматно-ориентированный метод синтаксического контроля диаграммных языков. Рассматриваются вопросы реализации системы. Изложение материала ведется на примере UML-диаграмм. Аналогичные результаты получены для IDEF, ARIS и SDL.

1. Требования к системе

Система обеспечивает следующие основные функции:

1. создание диаграмм графических языков;
2. добавление новых нотаций графических диаграммных языков;
3. контроль и анализ построенных диаграмм по предварительно загруженным в систему описаниям языка и алгоритмам анализа с диагностикой синтаксических и семантических ошибок;
4. добавление синтаксических и семантических правил графических языков.

К системе предъявляются следующие требования:

1. возможность независимой работы приложения, т.е. интеграция приложений, реализованных на произвольных языках (при условии возможности написания расширений для построителя диаграмм);
2. добавление нотации нового диаграммного языка должно происходить с наименьшими усилиями;
3. архитектура интеграционного решения должна быть основана на общепризнанных стандартах.

Первое требование обеспечивается реализацией Web-сервера по технологии REST. Используется гибко настраиваемый REST-сервер, написанный на языке C#, который может быть запущен на машине конечного пользователя, либо развернут на удаленном сервере в сети.

Второе требование реализуется с помощью автоматного программирования.

2. Автоматное программирование

Автоматное программирование – это парадигма программирования, при использовании которой программа или её фрагмент осмысливается как модель автомата [2].

В зависимости от конкретной задачи в автоматном программировании могут использоваться как конечные автоматы, так и автоматы более сложной структуры.

Разработка программы производится по секциям (функциям, процедурам), в соответствии с командой работы автомата «если текущее состояние и вход, то выход и переход в следующее состояние». Передача данных между командами осуществляется только через явно обозначенное множество переменных.

2.1. RV-грамматика

RV – грамматикой языка L(G) называется упорядоченная пятерка непустых множеств

$G = (V, \Sigma, \underline{\Sigma}, R, r_0)$ [3], где

$V = \{\nu_e, e = 1, L\}$ – вспомогательный алфавит;

$\Sigma = \{a_t, t = 1, T\}$ – терминальный алфавит графического языка;

$\underline{\Sigma} = \{a_t, t = 1, T\}$ – квазитерминальный алфавит;

$R = \{r_i, i = 0, I\}$ – схема грамматики G;

$r_0 \in R$ – аксиома RV-грамматики.

Продукция $P_{ij} \in r_i$ имеет вид P_{ij} : $\widetilde{a}_t \xrightarrow{\Omega_\mu[W_\nu(\gamma_1, \ldots, \gamma_n)]} r_m$, где

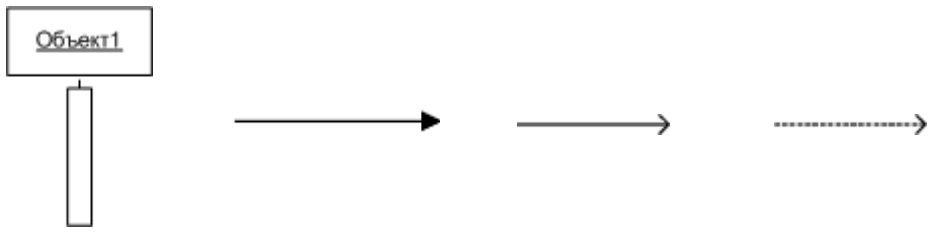
- $W_\nu(\gamma_1, \ldots, \gamma_n)$ – n – арное отношение, определяющее вид операции над внутренней памятью в зависимости от $\nu \in \{0, 1, 2, 3\}$;
- Ω_μ – оператор модификации, определенным образом изменяющий вид операции над памятью, причем $\mu \in \{0, 1, 2\}$;
- $r_m \in R$ – имя комплекса продукции – преемника.

Методика построения RV-грамматики состоит в следующем. На первом этапе определяются элементы, которые могут иметь более одного входа или выхода. Для связей, исходящих из этих объектов, определяются семантические различия. На втором этапе определяется квазитерминальный алфавит из терминального алфавита. На следующем этапе строится матрица допустимых паросочетаний. Затем определяются операции над внутренней памятью для определенных категорий квазитермов. На последнем этапе синтеза RV – грамматики строится ее табличное, графовое или линейное представление. Затем производится анализ полученной грамматики с целью минимизации и устранения недетерминированности и неопределенности.

Разработаны грамматики для пяти видов UML-диаграмм: Активности, Последовательности, Вариантов использования, Классов и Пакетов.

Рассмотрим диаграмму последовательности [4]. В терминальный алфавит входят все графические примитивы, которыми можно оперировать при построении диаграммы. В таблице 1 приведен терминальный алфавит и квазитерминальный алфавит диаграммы последовательности.

Таблица 1: Графические примитивы языка UML, диаграмма последовательности (а – объект, б – сообщение, в – асинхронное сообщение, г – сообщение о возврате)



В таблице 2 приведена RV-грамматика. Грамматика уже минимизирована для построения минимального автомата разбора. RV-отношение определяет операции с памятью, которые необходимо произвести при смене состояний автомата. Например, при смене состояния с r_1 на r_2 необходимо произвести операцию $W_1(m^{t(1)})$, что обозначает запись в стек объекта, из которого исходит связь. При переходе из состояния r_1 в конечное состояние под действием сигнала no_label выполняется операция *, которая обозначает проверку. Для

данной диаграммы корректным будет считаться переход в конечное состояние при пустом стеке.

Таблица 2: RV-грамматика для диаграммы последовательности

№ пп	Комплекс	Квази-терм	Комплекс – преемник	RV – отношение
1	r_0	OBJ	r_6	⊘
2	r_1	M	r_2	$W_1(m^{t(1)})$
3		M_A	r_2	$W_1(m^{t(1)})$
4		M_R	r_2	$W_2(m^{t(1)})$
5		no_label	r_k	*
6	r_2	OBJ	r_1	⊘

Реализован алгоритм нейтрализации ошибок, основанный на формировании множеств комплексов-продолжателей.

Определены множества синтаксических и семантических ошибок. Для 5 видов UML-диаграмм их 16 и 9 соответственно.

Для диаграммы последовательности список ошибок представлен ниже.

Классы синтаксических ошибок:

- Отсутствие связи
  - не рекурсивное пересечение фокусов управления
- Недопустимая связь
  - связь вызова направленная справа налево
  - связь возврата направленная слева направо
- Вход связи в связь
  - любой тип связи входящий в другой тип связи
- Вызов, направленный в линию жизни

Классы семантических ошибок:

- Переход к объекту без возврата

Выводы

В заключении необходимо отметить применимость предлагаемых грамматик для контроля неструктурированных языков и языков, содержащих параллелизм. RV – грамматики обладают минимальными затратами памяти и линейным временем разбора.

Литература

1. Афанасьев А.Н. и др. Контроль информации в системах автоматизации проектирования. // Саратов: Изд-во Саратовского университета, 1985.- 136 с.
2. Шалыто А.А. Технология автоматного программирования //Мир ПК. -2003. – № 10.
3. Шаров О.Г., Афанасьев А.Н. Синтаксически-ориентированная реализации графических языков на основе автоматных графических грамматик // Программирование. – 2005. – № 5. – С. 56-66.
4. М. Фаулер, К. Скотт. UML: Основы. – Пер. с англ.// СПБ: Символ-Плюс, 2002. – 192 с.

УДК 519.682.4, 371.32

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД И ОБЪЕКТНО-ОРИЕНТИРОВАННЫЕ ЯЗЫКИ: ПРОБЛЕМЫ ИЗУЧЕНИЯ В ВУЗЕ

Рудакова Анна Александровна, преподаватель, Кузбасский государственный технический университет филиал в г. Междуреченске, Россия, Междуреченск, anaruda@rambler.ru

Объектно-ориентированный подход позволяет разработать хорошо структурированные и достаточно просто видоизменяемые программные системы. Этим и объясняется интерес программистов к объектно-ориентированному подходу и объектно-ориентированным языкам программирования. Объектно-ориентированный подход является одним из наиболее популярных и усиленно развивающихся направлений теоретического и прикладного программирования. Он превратился в стандарт во многих фирмах и программистских сообществах. Во многих профессиональных системах программирования есть средства именно для поддержки объектно-ориентированного программирования. Эта технология вместе с организационными новшествами резко повысила качество программ, продуктивность программиста, эффективность коллективной работы программистов. Поэтому объектно-ориентированное программирование и объектно-ориентированная технология программирования уже стали неотъемлемой частью большинства вузовских учебных программ.

Формы и методы обучения объектно-ориентированному программированию и объектно-ориентированным технологиям во многом задаются целями обучения, содержанием курса и условиями преподавания. При изучении объектных технологий и объектно-ориентированных языков возникает ряд трудностей в освоении ключевых понятий и правил данной методологии. Какие проблемы возникают в учебном процессе ВУЗа по освоению этой технологии?

Одна из основных, на мой взгляд, проблем – педагогический состав. Учителя школы – люди старшего поколения (по последним исследованиям, проводимым по заказу Фонда специалистами МГУ, средний возраст учителей в России – 52 года, молодые учителя в школах не удерживаются). Они имеют огромный опыт в использовании технологии структурного проектирования программ, который основан на алгоритмической декомпозиции больших систем. В результате у школьника формируется стереотип процедурного мышления, вследствие чего переход к объектно-ориентированному программированию для них труден и сложен. Преподавателю ВУЗа приходится полностью менять уже сформированное системное мышление. Но данную проблему можно решить при организации изучения объектно-ориентированного программирования в самом начале учебного процесса: лучше, конечно, в школе, или на крайний случай – на первом курсе ВУЗа. В этом случае неизбежная перестройка профессионального мышления не будет помехой учебному процессу.

Информационные технологии являются одной из наиболее быстро развивающихся областей нынешней жизни. Общеизвестный факт, что новые технологии, проекты, названия и аббревиатуры появляются едва ли не каждый день. Поэтому знания приемов программирования, которые студенты получают на начальных курсах, уже к моменту окончания ВУЗа в большей мере себя изживают. Отсюда возникает еще одна проблема: первокурсник подготовлен профессиональнее, чем выпускник. Отсюда возникает парадокс: зачем и как преподавать объектно-ориентированного программирование и где гарантия, что через несколько лет не появятся свежие идеи и само объектное программирование канет в историю, как и канули идеи структурного программирования? Поэтому привитие молодому специалисту качеств быстрого и своевременного реагирования на новые идеи и технологии программирования – главная задача ВУЗа и преподавателя, прежде всего.

ООП предлагает новый мощный способ решения проблемы сложности программ. Вместо того, чтобы рассматривать программу как набор последовательно выполняемых инструкций, в ООП программа представляется в виде совокупности объектов, обладающих сходными свойствами и набором действий, которые можно с ними производить.[2] А сама модификация программ осуществляется только на этапе проектирования. Вследствие чего у студента отсутствует стимул к обеспечению модифицируемости программы (стиль его работы – «сдал и забыл»). Легкость получения работающей программы на этапе обучения не формирует основ для создания программных продуктов, которые соответствуют современным требованиям, отсутствует само понимание идеи, реализация принципов

объектно-ориентированного программирования и т.д. Программирование стало простым, студент, нахватав верхушки, считает себя профессионалом, сложнее простроить сам процесс обучения таким образом, чтобы не сформировать у него эту иллюзию.

В любом техническом ВУЗе обучение студентов программированию стоит в одном ряду с изучением большого числа других технических и специальных дисциплин. Дифференцированный подход между различными дисциплинами достиг большого размаха и не мог не затронуть объектно-ориентированное программирование. В любом ВУЗе освоение разных дисциплин осуществляется под руководством различных преподавателей, и у студентов возникают трудности в осознании целостности раздробленного по разным дисциплинам учебного материала.

Одной из проблем является оценка знаний и умений студентов в области объектно-ориентированного программирования. Как правило, студент, выполняя лабораторные и практические работы, решает простые задачи (большая их часть уже решена до него). А технология объектно-ориентированного программирования предполагает, прежде всего, создание сложных программ [5]. Преподаватель, упрощая картину, в «интересах самих же студентов», лишает их опыта создания сложных программ, а также не руководит процессом их разработки.

В процессе обучения студенты не имеют хороших примеров по реализации объектно-ориентированных программ (особенно идейно новых и свежих) и подробные комментарии по выбору решений фактически отсутствуют. Разработчики программных средств раскрывают только внешнюю логику программы, а реализация программы является коммерческой тайной. Преподаватель, предлагая примеры программ студентам, действует субъективно, имея под рукой либо то, что есть, либо к чему имеет пристрастие, а студенты находятся в зависимости от профессиональных увлечений самого преподавателя.

Сейчас стало доступно большое многообразие систем и языков объектно-ориентированного программирования, что осложняет не только их изучение, но и выбор самим преподавателем программных сред для обучения. Т.е. от предпочтения преподавателя зависит степень и специфика изучения, как прикладного программного обеспечения, так и систем программирования.

Также следует принять во внимание и традиционные для учебного процесса проблемы. Весь студенческий контингент неоднороден: кто-то пришел получить профессию, другие – просто за дипломом, третьи – по совету, а не по душе. Студент-внебюджетник – вообще контингент особый. И теперь перед преподавателем стоит вопрос выбора: на какого студента ориентироваться, кто платит (не секрет, что ВУЗ тоже зависит от студенческой платы, мало кто имеет федеральное финансирование), или на того, кто может, а главное, хочет учиться? Да и практические занятия перегоняют знания теории. Преподаватель должен правильно организовывать самостоятельную работу студентов уже после овладения им теоретических знаний. Последовательность дисциплин в процессе учебы также часто не отвечает требованиям первичности и вторичности приобретения знаний. А причина здесь – отнюдь не плохая организация учебного процесса, а тесная взаимосвязь разных дисциплин в рамках одной специальности. Поэтому преподавателю и приходиться отвлекаться на краткие экскурсы в другие дисциплины.

Учитывая все выше сказанное, какого же выпускника ВУЗа мы получаем? Студент, получил образование на примере составления простых программ, под руководством учителей и преподавателей, имеющих «структурно-ориентированную» технологию преподавания, и на примере уже успешных и отлаженных программ. И теперь на начальных этапах профессиональной деятельности этот выпускник должен самостоятельно отфильтровывать все идеи структурного программирования программ, которые никак не совместимы с идеями объектно-ориентированной технологии.

А работодатели в области программной индустрии предъявляют к соискателю должности иные требования: возможность самостоятельно реализовать программный проект. Значит, задача ВУЗа – создать студенту условия для получения за время обучения

«самостоятельно реализованного программного проекта». Поэтому все лабораторные, курсовые и дипломные работы, магистерские диссертации должны быть ориентированы, прежде всего, на предоставление студенту такой возможности.

Изучение программирования – одновременно и простое (относительно), и сложное, а учебный процесс по изучению объектно-ориентированных технологий накопил целый ряд противоречий объективного и субъективного характера.

Литература

1. Савчук И. Почему объектно-ориентированное программирование провалилось? http://citforum.ru/gazeta/165/
2. Роберт Лафоре, Объектно-ориентированное программирование в С++. – Питер, 2008. – 923 с.
3. Лобачев А.А, Куликова О.В. Выбор языка для обучения программированию / Информационные технологии в образовании. XVIII Международная конференция-выставка: Сборник трудов участников конференции. Ч. VI. – М.: МИФИ, 2008.
4. Буч Г. Объектно-ориентированный анализ и проектирование. – СПб.: Бином, 1998.
5. Буч Г. Объектно-ориентированный анализ и проектирование приложений на Си++. –М.: Бином; СПб.: Невский диалект, 2001. -560с.

УДК 004.4

СОЗДАНИЕ МОДЕЛИ ХРАНИЛИЩА ДАННЫХ КОРПОРАТИВНОЙ ИНФОРМАЦИОННО-АНАЛИТИЧЕСКОЙ СИСТЕМЫ ПРЕДПРИЯТИИ

Лучкин Николай Анатольевич, аспирант, Омский государственный технический университет, Россия, Омск, nikolay.luchkin@gmail.com

Введение

К настоящему времени во многих организациях накоплены колоссальные объемы данных, на основе которых можно решать самые разнообразные аналитические и управленческие задачи в любой сфере деятельности. Проблемы хранения и обработки аналитической информации становятся все более актуальными и привлекают внимание специалистов и фирм, работающих в области информационных технологий.

В идеале работа аналитиков и руководителей различных уровней должна быть организована так, чтобы они могли иметь доступ ко всей интересующей их информации, а также пользоваться удобными и простыми средствами представления и работы с этой информацией. Именно на достижение этих целей и направлены информационные технологии, объединяющиеся под общим названием хранилища данных.

Для предоставления необходимой для принятия решений информации обычно приходится собирать данные из нескольких транзакционных баз данных различной структуры и содержания. Основная проблема при этом состоит в несогласованности и противоречивости этих баз-источников, отсутствии единого логического взгляда на корпоративные данные. Решением этой проблемы является хранилище данных. В основе концепции хранилищ данных лежит важная идея интеграции ранее разъединенных детализированных данных, содержащихся в исторических архивах, накапливаемых в традиционных системах, поступающих из внешних источников, в единую базу данных.

В общем виде архитектура корпоративной информационно-аналитической системы описывается схемой с тремя выделенными слоями (рис.1): извлечение, преобразование и загрузка данных; хранение данных; анализ данных (рабочие места пользователей).

Технология функционирования системы заключается в следующем. Данные поступают из различных внутренних транзакционных систем, от подчиненных структур, от внешних организаций в соответствии с установленным регламентом, формами и макетами отчетности. Вся эта информация проверяется, согласуется, преобразуется и помещается в хранилище и витрины данных. После этого пользователи с помощью специализированных

инструментальных средств получают необходимую им информацию для построения различных табличных и графических представлений, прогнозирования, моделирования и выполнения других аналитических задач. В следующих разделах рассмотрим подробнее каждый из слоев.

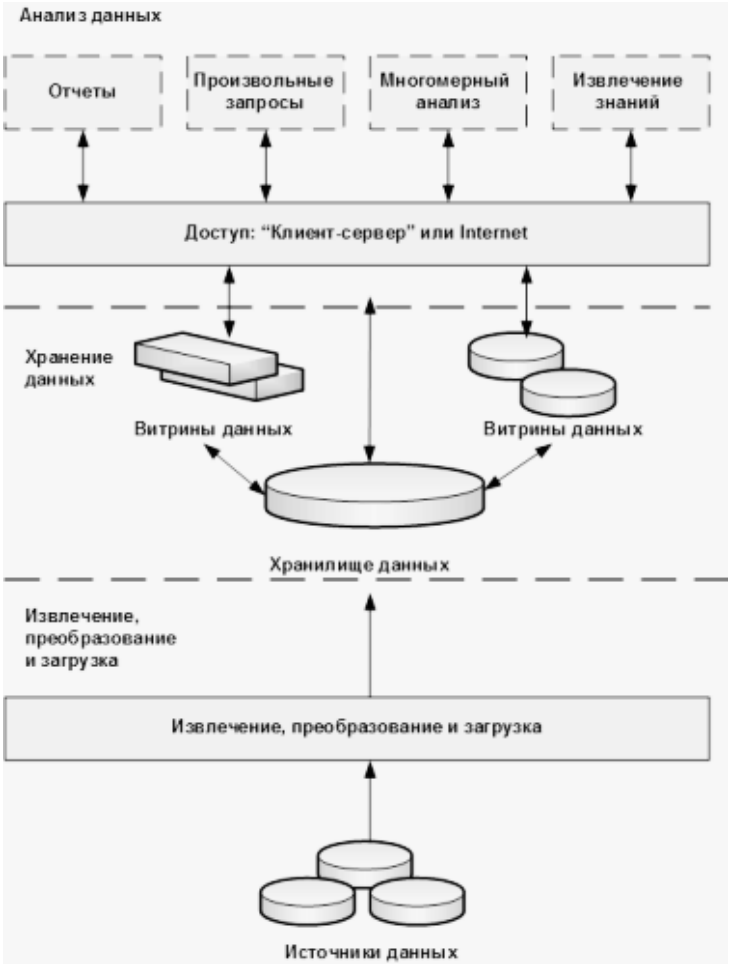


Рис. 1 – Архитектура корпоративной информационно-аналитической системы

1. Извлечение, преобразование и загрузка файлов

В качестве источников информации для хранилища могут использоваться базы данных внутренних транзакционных систем, информационные системы подчиненных организаций, данные, поступающие из внешних организаций.

С организационной точки зрения, данный слой включает подразделения и структуры организации всех уровней, поддерживающие базы данных оперативного доступа. Он представляет собой низовой уровень генерации информации, уровень внутренних и внешних информационных источников, вырабатывающих "сырую" информацию. Эта информация является рабочей для повседневной деятельности различных подразделений, которые ее вырабатывают и используют.

С системно-технической точки зрения данный слой представлен ЛВС всех подразделений всех уровней, к которым подключены специализированные технические комплексы, хранящие информацию. В качестве таких технических комплексов могут выступать: серверы реляционных баз данных на базе компьютеров под управлением Windows NT, Unix и др.; файловые серверы, на которых установлена какая-либо система обработки данных или сетевая версия СУБД; персональные компьютеры с локальными персональными базами данных или файлами. Из источников данных информация перемещается на основе некоторого регламента в централизованное хранилище. Как правило, необходимые для хранилища данные не хранятся в окончательном виде ни в одной из транзакционных систем.

Кроме того, несмотря на различную функциональную направленность, исходные транзакционные системы часто «пересекаются» по данным, т.е. их локальные базы данных содержат однотипную по смыслу информацию. Это касается нормативно-справочной информации, которая используется в том или ином виде в любой оперативной системе. Перед загрузкой в хранилище вся эта информация должна быть согласована, чтобы обеспечить целостность и непротиворечивость аналитических данных.

Согласование данных необходимо и при загрузке данных из одного источника. В хранилище хранятся исторические данные, т.е. данные за достаточно большой промежуток времени. В оперативной системе данные хранятся в целостном виде за ограниченный промежуток, после чего они отправляются в архив. При изменениях в структуре или собственно данных архивы не подвергаются никакой дополнительной обработке, а хранятся в исходном виде.

Таким образом, загрузка данных из источников в хранилище осуществляется специальными процедурами, позволяющими:

- извлекать данные из различных баз данных, текстовых файлов;
- выполнять различные типы согласования и очистки данных;
- преобразовывать данные при перемещении их от источников к хранилищу;
- загружать согласованные и «очищенные» данные в структуры хранилища.

Для разработки, поддержки и выполнения таких процедур рекомендуется использовать специализированный инструментарий, предназначенный для автоматизации процессов извлечения данных из источников, их преобразования и загрузки в целевое хранилище.

Извлечение, преобразование и загрузка данных должна осуществляться как непосредственно вызовом соответствующих процедур, так и в автоматическом режиме, на основе «скриптов» и расписаний, составленных на этапе разработки системы.

2. Хранение данных

Второй слой предназначен непосредственно для хранения проверенной, согласованной, непротиворечивой и хронологически целостной информации, которую с достаточно высокой степенью уверенности можно считать достоверной.

Хранилище данных не ориентировано на решение какой-либо определенной функциональной аналитической задачи. Цель хранилища – обеспечить целостность и поддерживать хронологию всевозможных корпоративных данных, и с этой точки зрения оно нейтрально по отношению к приложениям. В связи с этим, в большинстве случаев для выполнения определенного комплекса функционально замкнутых аналитических задач рационально создавать витрины данных, в основе которых может быть как многомерная, так и реляционная модель данных. По существу витрина представляет собой относительно небольшое функционально-ориентированное хранилище, в котором информация хранится специальным образом, оптимизированным с точки зрения решения конкретных аналитических задач некоторого подразделения или группы аналитиков.

Информация попадает в витрины из хранилища, и в этом случае витрины называются зависимыми. Возможна также ситуация, когда источником информации для пополнения витрин служат непосредственно оперативные и внешние транзакционные системы. Такие витрины, получившие название независимых, как правило, рассматриваются как временное решение, позволяющее достаточно быстро и с небольшими затратами решить наиболее важные задачи, оценить преимущества нового подхода, сформулировать некоторые рекомендации для более масштабного проекта разработки общего хранилища.

Хранилище реализуется в виде реляционной базы данных, работающей под управлением мощной реляционной СУБД. Такая СУБД должна поддерживать эффективную работу с терабайтными объемами информации, иметь развитые средства ограничения доступа, обеспечивать повышенный уровень надежности и секретности, соответствовать необходимым требованиям по восстановлению и архивации. Витрины данных могут строиться на основе как реляционной, так и многомерной технологии баз данных. Для

достаточно большой части аналитических приложений оказывается удобной и эффективной технология интерактивного многомерного анализа, и в этом случае витрина представляет собой многомерную базу данных, реализованную в архитектуре OLAP, ROLAP или HOLAP.

3. Анализ данных

Для организации доступа аналитиков к данным хранилища и витрин используются специализированные рабочие места, поддерживающие необходимые технологии как оперативного, так и долговременного анализа. Результаты работы аналитиков оформляются в виде отчетов, графиков, рекомендаций и сохраняются как на локальном компьютере, так и в общедоступном узле локальной сети.

Аналитическая деятельность в рамках корпорации достаточно разнообразна и определяется характером решаемых задач, организационными особенностями компании, уровнем и степенью подготовленности аналитиков.

В связи с этим современный подход к инструментальным средствам анализа не ограничивается использованием какой-то одной технологии. В настоящее время принято различать четыре основных вида аналитической деятельности: стандартная отчетность, нерегламентированные запросы, многомерный анализ (OLAP) и извлечение знаний.

Каждая из этих технологий имеет свои особенности, определенный набор типовых задач и должна поддерживаться специализированной инструментальной средой.

Базы данных будут продолжать развиваться, а объемы информации в компьютерах – увеличиваться. Усложнение производственных процессов, "интеллектуализация" контрольно-измерительных приборов, датчиков и исполнительных механизмов, требования конечного пользователя относительно повышения объемов и качества информации делают это высказывание особенно справедливым для промышленных условий.

Выполнять проектирование модели хранилища данных на основе корпоративной модели данных организации целесообразно с помощью CASE-инструментария. Использование CASE-инструментария увеличивает производительность труда проектировщика хранилища данных, особенно это касается в случае средних и крупных проектов.

Литература

1. Архипенков С., Голубев Д., Максименко О. Хранилища данных. От концепции до внедрения М.: Диалог-МИФИ, 2002. – 528 с.
2. Архипенков С. Аналитические системы на базе ORACLE Express OLAP. Проектирование, создание, сопровождение М.: Диалог-МИФИ, 2000. – 320 с.
3. Кузнецов С. Д. Основы баз данных. – 2-е изд. – М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007. – 484 с.
4. Кузнецов С., Артемьев В. Обзор возможностей применения ведущих СУБД для построения хранилищ данных (DataWarehouse).

УДК 004.457

АРХИТЕКТУРА АВТОМАТИЧЕСКОЙ СИСТЕМЫ ОБНАРУЖЕНИЯ И ПРЕДОТВРАЩЕНИЯ АТАК

Крылов Александр Юрьевич, студент 3-го курса, Ивановский государственный химико-технологический университет, Россия, Иваново, Krylov_Al@mail.ru

Галиаскаров Эдуард Геннадьевич, к.х.н., доцент, доцент, Ивановский государственный химико-технологический университет, Россия, Иваново, galiaskarov@isuct.ru

Введение

Система обнаружения вторжений (СОВ) – программное или аппаратное средство, предназначенное для выявления фактов неавторизованного доступа в компьютерную

систему или сеть, либо несанкционированного управления ими, в основном через Интернет. Системы обнаружения вторжений используются для обнаружения некоторых типов вредоносной активности, которая может нарушить безопасность компьютерной системы. К такой активности относятся сетевые атаки против уязвимых сервисов, атаки, направленные на повышение привилегий, неавторизованный доступ к важным файлам, а также действия вредоносного программного обеспечения (компьютерных вирусов, троянов и червей) [1].

В связи с обилием на рынке программного обеспечения различных систем выявления и предотвращения атак от различных производителей появляются самые разные методы и технологии работы подобных систем, что требует специфического подхода к разработке программной архитектуры решения. Именно поэтому совершенствование и расширение некоторых из них удается довольно легко, а для других – это довольно сложная и трудоемкая задача. Поэтому в некоторых случаях разработанная программная архитектура лишь отчасти оправдывает себя. Исходя из этого, целью нашей работы является разработка такой архитектуры системы, которая будет поддерживать любые методы (или, по крайней мере, большую часть) выявления и предотвращения атак. Такая архитектура могла бы выступать своего рода «шаблоном» для реализации подобных систем.

Разрабатываемая архитектура систем выявления и предотвращения атак, по нашему мнению, должна удовлетворять следующим условиям:

- обеспечивать универсальность решения, т.е. поддерживать возможность использования различных методов, технологий и стратегий выявления злоумышленных действий [2, 3];
- обеспечивать гибкость использования, т.е. позволять администратору системы (либо самой системе при наличии возможности самообучения) вносить описание новых видов атак в библиотеку описаний атак системы;
- позволять быстро и эффективно реализовывать какую-либо специфическую, однонаправленную систему выявления и предотвращения сетевых атак, специального назначения;
- поддерживать возможность распределения функциональных компонентов системы по нескольким компьютерам в сети;
- обеспечивать возможность замены какого-либо функционального блока системы, не затрагивая остальные.

Все вышеописанные требования к архитектуре программного средства являются наиболее важными и при проектировании рассматривались в первую очередь. В итоге, разработав модель такого решения, мы вывели следующие концепции (рис. 1):

- вся система подразделяется на функциональные блоки, каждый из которых выполняет лишь свои определенные функции;
- коммуникация между функциональными блоками обеспечивается при помощи программных интерфейсов, предоставляемых и используемых самими компонентами системы;
- с самой высокоуровневой точки зрения система имеет четко выраженную клиент-серверную архитектуру, в которой вся система разделяется на сетевых агентов, устанавливаемых на защищаемых компьютерах пользователей (клиенты), и серверную часть системы (сервер), обеспечивающую принятие решений и разработку плана реагирования. При этом сама серверная часть также в свою очередь может быть разбита на некоторые функциональные компоненты (подсистемы).

Реализация системы, удовлетворяющей выделенным требованиям, представлена на рисунке 1.

Рис. 1 – Возможный вариант размещения компонентов системы обнаружения и предотвращения атак

1. Описание функциональных компонентов системы

Сетевые агенты. Являются модулями, располагаемыми на компьютерах сети. Основная задача состоит в отслеживании на этих компьютерах подозрительных действий, оповещении системы о них и выполнении соответствующих инструкций от серверной части системы: управляющих, конфигурационных и направленных на предотвращение действий злоумышленников. Исходя из этого, устанавливаются на компьютерах в сети, запускаясь от имени администратора и функционируя в фоновом режиме в виде сервиса, т.о. не создавая помех работе пользователя данного защищаемого от действий злоумышленников компьютера.

Подсистема отслеживания событий. Обеспечивает конфигурирование сетевых агентов и получение оповещений о происходящих на компьютерах сети подозрительных действиях. Получив оповещение о каком-либо событии, произошедшем на одном из компьютеров сети, подсистема организует очередь, в соответствии с которой, используя интерфейс инициализации процесса обработки события, предоставляемого подсистемой реагирования на событие, передает ей это сообщение (в этой системе сообщение передается блоку расшифровки сообщения), тем самым инициализируя процесс расшифровки и представления на «языке системы» информации о событии.

Подсистема реагирования на событие: Подсистема расшифровки сообщения. Задачей этого компонента системы является расшифровка полученной информации о произошедшем на каком-либо из компьютеров сети событии и представление ее в удобном для принятия решения виде. Этот вид зависит от подсистемы принятия решений, а, следовательно, напрямую зависит от технологии выявления атак, используемых в системе. Собрав полученную информацию в таком виде, подсистема обработки событий, используя интерфейс, предоставляемый подсистемой принятия решений, производит запрос к этому компоненту, передавая ему всю необходимую для принятия решения информацию, тем самым инициализируя процесс принятия решения.

Подсистема реагирования на событие: Подсистема принятия решений. Данный компонент является функциональным ядром системы, обеспечивая принятие решения – является ли происходящее атакой или нет (в крайних случаях может быть реализовано и принятие других решений). Стратегии и технологии принятия решения могут быть самыми разными, но реализуются лишь в этом компоненте, предоставляющем интерфейс для запросов принятия решений. Всю необходимую для обеспечения процесса принятия

решения информацию компонент получает из подсистемы доступа к информации об определениях атак, используя соответствующий интерфейс. Для выдачи результата процесса принятия решения компонент использует интерфейс выдачи решения, предоставляемый подсистемой разработки плана действий, тем самым инициализируя этот процесс в соответствующем блоке.

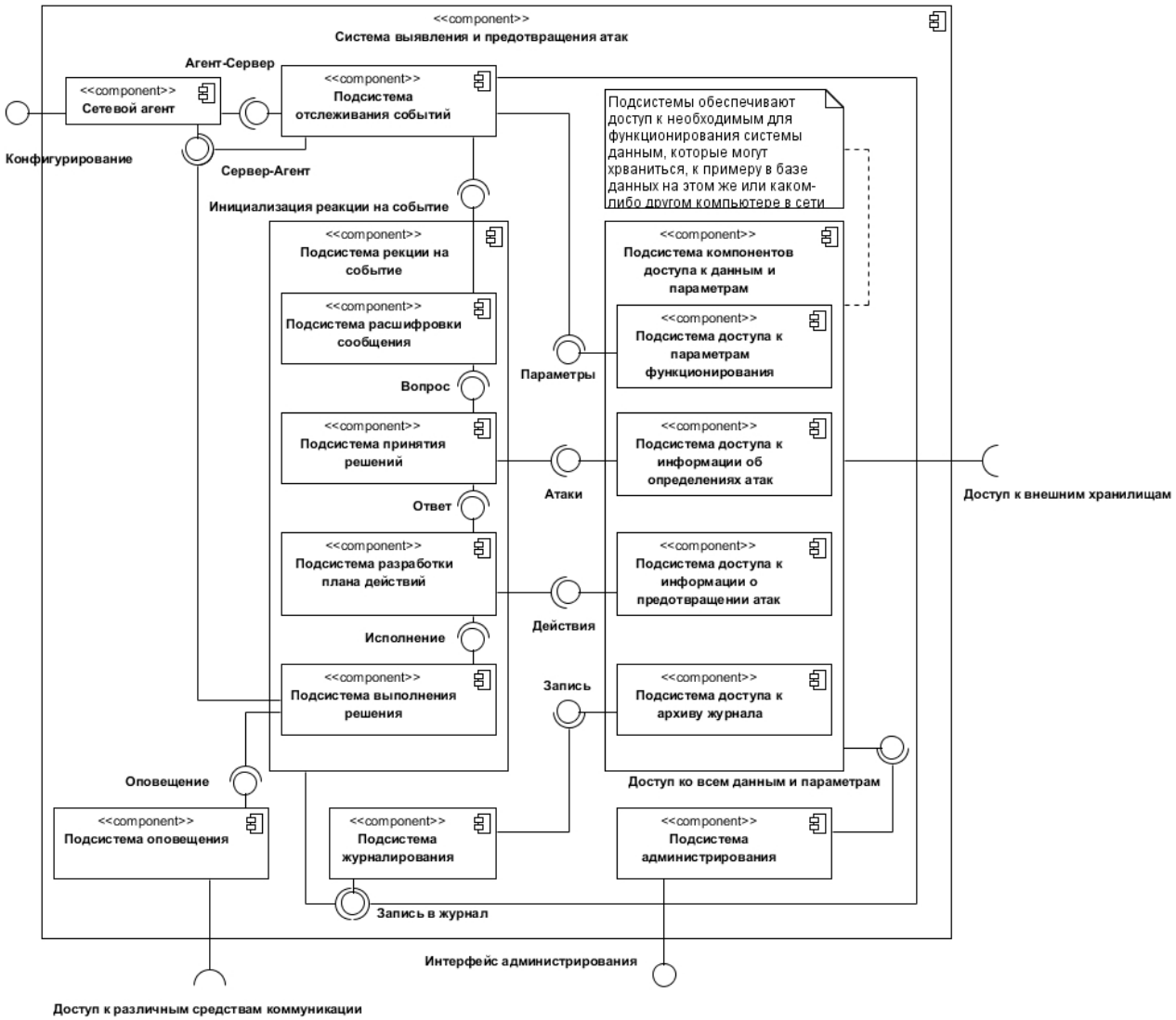


Рис. 2 – Компонентная структура системы обнаружения и предотвращения атак

Подсистема реагирования на событие: Подсистема разработки плана действий. Данный компонент, в зависимости от решения, полученного от подсистемы принятия решений, и используя подсистему доступа к информации о предотвращении атак, решает, какие действия должны быть произведены. Целью его функционирования является составление алгоритма действий как реакции на произошедшее событие. В разработанный план действий входят инструкции сетевым агентам для последующего исполнения ими этих инструкций, задачи оповещения администратора системы и сообщения для подсистемы журналирования. Готовый план передается подсистеме выполнения решений по предоставляемому ею интерфейсу, тем самым инициализируя процесс выполнения разработанного плана действий.

Подсистема реагирования на событие: Подсистема выполнения решения. Иначе данный компонент можно назвать диспетчером исполнения решения: в его задачи входят распределение обязанностей из полученного плана действий по другим функциональным компонентам системы, так или иначе связанных с процессом реагирования на происходящее событие, используя предоставляемые этими компонентами и доступные данной подсистеме интерфейсы. Выполняя решение, подсистема отправляет инструкции по реагированию на

произошедшее событие сетевым агентам, связанным с этим событием, запускает процесс оповещения администратора системы, собирает и передает подсистеме журналирования необходимые для записи в журнал работы системы данные. При этом одной из основных задач подсистемы является отслеживание выполнения поставленных задач и выполнение определенных действий в случае ошибок, возникших в процессе исполнения составленного плана реагирования.

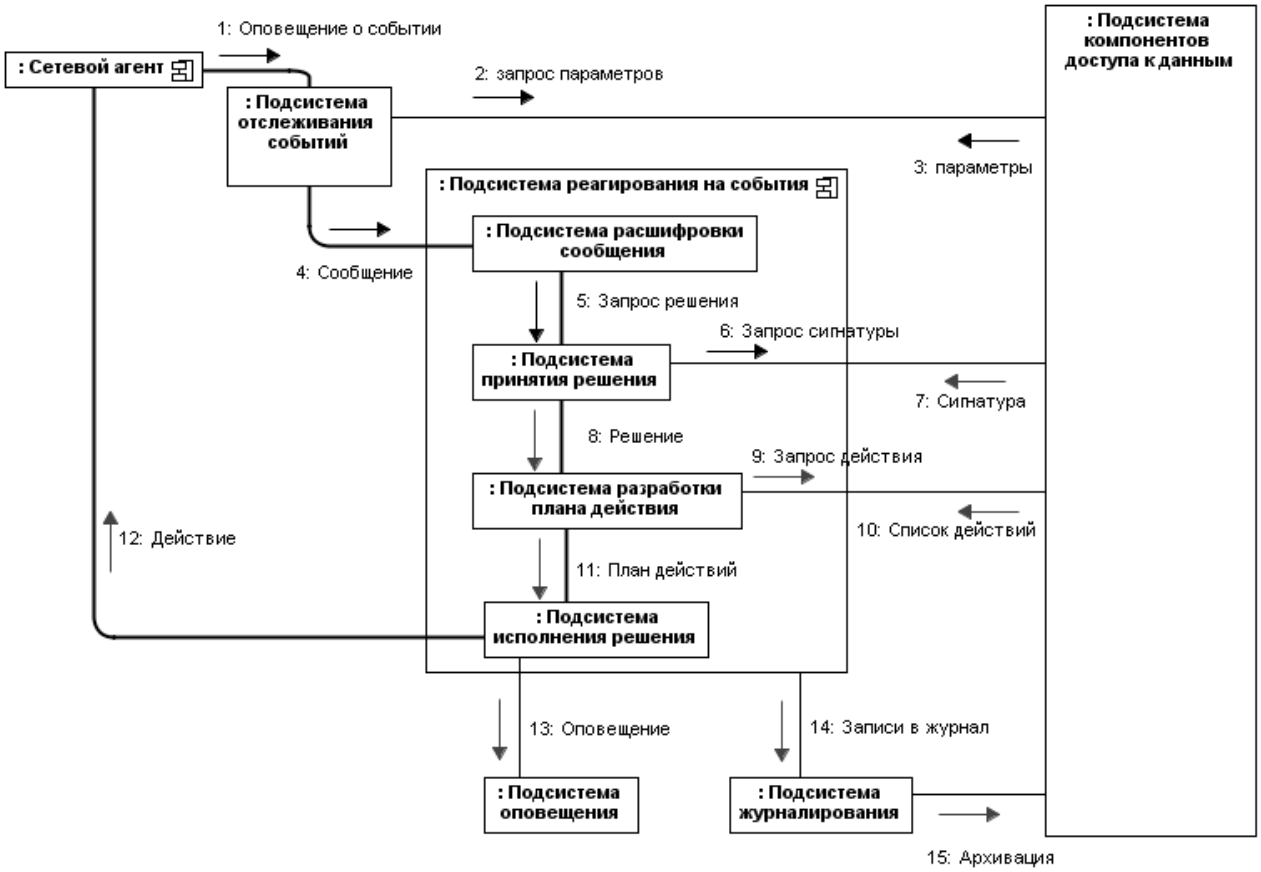


Рис. 3 – Типичный сценарий реакции системы на подозрительное событие

Подсистема журналирования. Целью функционирования этого компонента системы является администрирование журнала работы системы в целом, регистрация в нем всего, что важно для последующего анализа данные: произошедшие события, принятые решения, разработанные планы реагирования, результаты процесса исполнения, ошибки и т.д. Вид журнала работы системы и регистрируемые в нем данные зависят от настроек системы, что позволяет при необходимости отключить выбранный компонент вовсе. При этом компонент предоставляет единый для всех связанных с ним компонентов системы интерфейс, по которому принимает регистрируемые данные. Журнал работы системы, используемый данным компонентом, хранится в архиве подсистемы журналирования, доступ к которому осуществляется при помощи соответствующего модуля подсистемы компонентов доступа к данным и параметрам.

Подсистема оповещения. Отвечая за оповещение выбранных в зависимости от настроек системы лиц о произошедшем событии и реакции на него, компонент системы выполняет все необходимые действия для связи с доступными для этого средствами. Как и подсистема журналирования, обеспечивает лишь один интерфейс, по которому принимает информацию, которую необходимо передать лицам, оповещаемым в данном случае.

Подсистема компонентов доступа к данным и параметрам. Предназначается для получения и выдачи при запросе таких данных, как информация об определениях атак (например, сигнатуры), информация о предотвращения атак, конфигурационные данные, архив работы системы. Таким образом, подсистемы данного компонента являются

интерфейсами к соответствующим базам знаний системы, которые могут быть реализованы самыми разными методами – конфигурационными файлами, базами данных и др. Для поддержки возможности использования внешних по отношению к системе хранилищ данных подсистема имеет возможность использовать внешние интерфейсы, предоставляемые этими хранилищами.

Подсистема администрирования. Консоль администратора системы. Выполняет функции интерфейса (в том числе и графического) между системой и ее администратором, принимая для установки параметры функционирования и выдавая их по запросу. Является необязательным компонентом системы, если она позиционируется как автоматическая.

На рис. 3 представлен типичный сценарий реакции системы на возникшее подозрительное событие. Сетевой агент в режиме реального времени оповещает систему реагирования о возникшем подозрительном событии. Подсистема отслеживания событий регистрирует сообщение, ставит его в очередь на обработку и передает очередное сообщение в подсистему реагирования. Подсистема расшифровки сообщения в соответствии с заложенными в нее правилами анализирует сообщение и преобразует его в вид, удобный для использования подсистемой принятия решения. Подсистема принятия решения анализирует информацию в сообщении, сверяясь с банком знаний (например, с банком известных сигнатур) и принимает решение, т.е. по сути, передает управляющую команду на организацию защитных действий в подсистему разработки плана действий. Подсистема планирования формирует необходимый алгоритм действий, который и передает в подсистему выполнения решения. Подсистема выполнения решения передает команды исполнительным модулям сетевого агента, одновременно инициируя передачу оповещения в соответствии с заданной программой. На протяжении всего цикла обработки подозрительного события подсистема реагирования на события производит подробное логирование своих действий и передает записи в подсистему журналирования.

Выводы

Предложенная архитектура предполагает наличие таких преимуществ от ее использования, как возможность выбирать или даже самостоятельно разрабатывать такие моменты реализации, как:

- вид интерфейсов системы: программные API, сетевые, радиоканалы, световые и др.; архитектура системы универсальна и не накладывает каких-либо ограничения на реализацию интерфейсов, кроме описанных выше;
- расположение компонентов системы: сосредоточение всех компонентов на одном компьютере в сети или распределение по нескольким. При этом нет необходимости располагать все компоненты в одной локальной сети – любой из них можно вынести за ее пределы;
- реализация компонентов системы: т.к. функционально система разбита на подсистемы, связанные описанными выше интерфейсами, нет необходимости реализации всей системы на каком-либо одном языке программирования. Это дает возможность использования разных платформ для реализации различных компонентов системы – здесь важна проработка интерфейсов и протоколов взаимодействия блоков. При этом каждый блок может быть реализован по-разному: в виде процесса, сервиса, даже в виде web-службы, интерфейс к которой будет обеспечиваться по протоколу http.

Исходя из полученных преимуществ разработанной архитектуры, можно утверждать, что предложенная архитектура является достаточно универсальной и не зависит от специфики среды применения, платформы и вида отслеживаемых атак.

Авторы выражают благодарность за полезные дискуссии Денису Юрьевичу Иванову, одному из авторов книги «Моделирование на UML».

Литература

1. Система обнаружения вторжений. [Электронный ресурс]:[материал из Википедии – свободной энциклопедии]. – Режим доступа: http://ru.wikipedia.org/wiki/IDS.
2. Д. Г. Колесников. Компьютерные атаки и технологии их обнаружения. [Электронный ресурс]:[Учебное электронное пособие]. – Режим доступа: http://protect.htmlweb.ru/attack.htm.
3. А. В. Полещук. Особенности построения комплексной системы информационной безопасности газотранспортного предприятия. Журнал «Information Security/Информационная безопасность», 2008. [Электронный ресурс]:[Электронная версия журнала Сайт ITSec.Ru-2008]. – Режим доступа: http://www.itsec.ru/articles2/Inf_security/mostransgaz-poleschuk.

УДК 004.4

БЕСПЛАТНЫЕ ИНСТРУМЕНТЫ ДЛЯ РАБОТЫ IT-АНАЛИТИКА

Сафин Павел Владимирович, магистрант, Самарский государственный аэрокосмический университет им. С.П. Королёва (Национальный исследовательский университет); бизнес-аналитик, «Интеллектуальные Системы (i-Sys)», Россия, Самара, pavel.safin@gmail.com

Проблема выбора аналитиком инструментов для продуктивной работы является основополагающей при работе над теми проектами, в которых нецелесообразно использовать систему управления требованиями, либо в случае отсутствия таковой. Данная статья отражает практический опыт автора и является обзором бесплатных инструментов визуального моделирования, которые аналитик может использовать в процессе разработки программного обеспечения при работе над небольшими проектами, а также в случае отсутствия системы управления требованиями, нецелесообразности её применения, либо при отсутствии навыков её использования. В то же время данная статья носит рекомендательный характер для использования упомянутых в статье бесплатных средств в учебных программах вузов, обучающих специалистов в области информационных технологий.

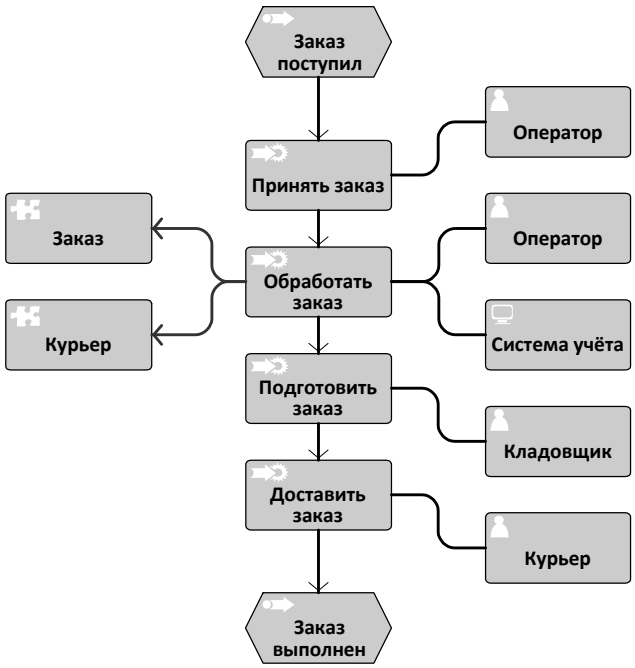


Рис. 4 – Бизнес-процесс выполнения заказа

Известно, что процесс жизненного цикла программного обеспечения начинается с фазы обследования предметной области. При обследовании предметной области необходимо выделить: основные и вспомогательные бизнес-процессы, потоки данных, сопровождающие каждый из процессов, а также лиц, ответственных за положительный итог процесса и выполнение функций, составляющих процесс. Для выполнения данных задач логичным выходом является использование бесплатного средства моделирования бизнес-процессов,

которым является Aris Express [1]. Процессы, описанные графически в документе обследования предметной области, помогают всей команде проекта вникнуть в саму суть проблемы, изучить её с точки зрения заказчика и пользователей будущей системы. Одновременно с описанием процессов можно проводить и выделение сущностей с целью разработки концептуальной схемы базы данных будущей системы. Концептуальную схему можно спроектировать в бесплатной версии ERWin, но с существенными ограничениями в 25 сущностей [2]. Пример смоделированного с помощью Aris Express бизнес-процесса представлен на рисунке 1. В качестве примера выбрана простейшая система учёта заказов.

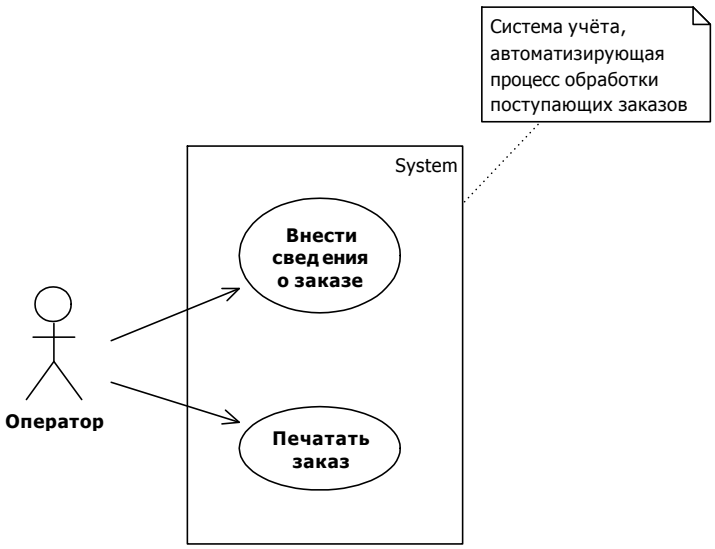


Рис. 5 – Использование системы учёта заказов

Параллельно с обследованием предметной области, как правило, формируются и документируются различные виды требований к будущей системе, одними из которых являются требования к использованию, структуре и поведению системы [3], список заинтересованных лиц и их цели. По мнению автора, наиболее привлекательным с точки зрения описания пользовательских требований является применение методики, предложенной Алистером Коберном, суть которой – выделение пользовательских целей и описание путей их достижения с помощью сценариев, описывающих диалог «пользователь – система» [4]. На данном этапе, кроме текстового описания шагов использования, понятного всем участникам проекта разработки, необходимо предложить и графическое представление использования всей системы. Делать это необходимо не только «ради того, чтоб было», а для того, чтобы в будущем вмиг вспомнить о том, какой же функциональностью обладает система. Лучшим способом справиться с поставленной задачей может помочь бесплатное средство визуального моделирования под названием StarUML, разработанное на Delphi [5]. Выполнив описание использования системы в StarUML, можно выиграть время на составлении спецификации использования системы, так как данное средство поддерживает и автоматизированную генерацию документации на основе описания вариантов использования, которое рекомендуется делать при создании данных элементов в этом программном средстве. В итоге выполнения работы по этапу с помощью StarUML можно получить готовый документ спецификации требований, а также визуальное изображение использования, структуры и поведения разрабатываемой системы в виде диаграмм UML, которые в дальнейшем можно использовать в проектной документации, например, в техническом задании, документе спецификации требований и пр. Визуальное изображение диаграммы использования для системы учёта заказов приведено на рисунке 2.

Одним из важных этапов создания новой информационной системы является разработка прототипов пользовательского интерфейса, помогающих разработчику понять то, какой элемент и где должен располагаться в пользовательском интерфейсе. При разработке

макетов можно руководствоваться как собственным опытом, так и использовать советы от признанных практиков [6]. Итоговые прототипы обычно формируются в едином документе функционального дизайна системы. Известно множество средств моделирования пользовательского интерфейса, но автор останавливает своё внимание на бесплатном приложении Pencil. Прототип пользовательского интерфейса для функции учёта поступающих заказов представлен на рисунке 3.

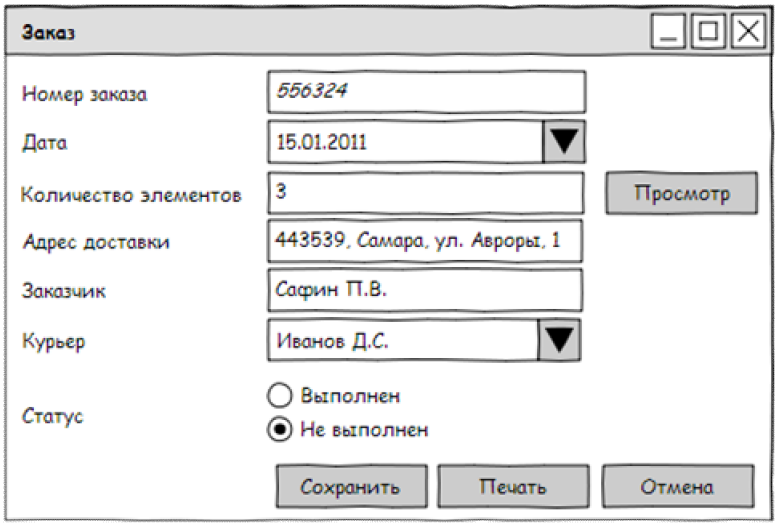


Рис. 6 – Прототип формы ввода информации о заказе системы учёта

Таким образом, в данной статье приведён обзор бесплатных средств визуального моделирования, которые могут упростить работу начинающего IT-аналитика в небольших и непродолжительных проектах. Описанные программные средства могут быть рекомендованы для изучения и практического применения в высших учебных заведениях, обучающих специалистов по направлениям, связанными с информационными технологиями.

Литература

1. http://www.ariscommunity.com/aris-express
2. http://erwin.com/products/detail/ca_erwin_data_modeler_community_edition/
3. 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).
4. Коберн А. Современные методы описания функциональных требований к системам. – М.: Лори, 2011. – ISBN 0-201-70225-8, ISBN 5-85582-152-8
5. http://staruml.sourceforge.net/en/
6. Влад В. Головач. Дизайн пользовательского интерфейса[2]. Искусство мыть слона. – 2009. – 94 с., http://uibook2.usethics.ru/uibookII.pdf
7. http://pencil.evolus.vn

УДК 004.4'2

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ КАРКАС ДЛЯ ПРЕДМЕТНОЙ ОБЛАСТИ ИГРОВОГО ЭЛЕКТРОННОГО ОБУЧЕНИЯ НА ОСНОВЕ РАЗВИВАЮЩИХ «ВОПРОС-ОТВЕТНЫХ» ЛИНГВИСТИЧЕСКИХ ЗАДАЧ[1]

Гусс Святослав Владимирович, аспирант/ассистент, Омский Государственный университет им. Ф.М. Достоевского, Россия, Омск, InfoGuss@Gmail.com

Введение

В предлагаемой статье представлено описание работы с объектно-ориентированным

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

каркасом поддержки занятий лингвистической направленности в игровой форме. Цель – продемонстрировать результаты использования технологий объектно-ориентированного проектирования в рамках создания и детализации объектно-ориентированного каркаса (object-oriented framework), расширяемого программного элемента повторного использования.

1. Каркас программных компонентов поддержки занятий лингвистической направленности

Каркас – «набор взаимодействующих классов, составляющих повторно-используемый дизайн для конкретного класса программ» [1]. Каркас программных компонентов поддержки занятий лингвистической направленности в игровой форме, предлагаемый вашему вниманию, в своём составе имеет набор взаимосвязанных и взаимодействующих классов предметной области игрового электронного обучения, на основе развивающих «вопрос-ответных» лингвистических задач. Подробнее о проблеме образования с помощью лингвистических задач и их представлением в форме игры можно ознакомиться в работах [2], [3]. О деталях внутреннего устройства и функционирования каркаса можно узнать в [4], [5]. О том, зачем использовать всё это в процессе разработки приложений, описано в [6], [7]. В настоящей работе предлагается описание того, как использовать механизмы, применимые в рамках объектно-ориентированного конструирования, для создания функционирующих компонентов на основе каркаса. Классы каркаса представлены на рис. 1.

Каркас хорошо подходит для создания компонентов игровых лингвистических задач типа «вопрос-ответ». В нём содержится ряд элементов, реализующих концепции предметной области, перечисленных ниже.

Язык (LanguageOfParticipants).

Алфавит (Alphabet) – набор литер выбранного языка.

Лексемы – элементы словаря (KnowledgeHolder.Vocabulary), могут быть отфильтрованы по длине, либо начальной литере.

Роль – определённое представление поведения в процессе решения задачи. Виды ролей: соперник (CompetitorRole), игрок (PlayerRole), судья (UmpireRole). Действия, производимые такими методами класса GameManager, как StartGame() и MakeGameStep() происходят от лица игрока.

Задача-головоломка (WordPuzzle). Состоит из обязательного условия, начального условия и секрета. **Обязательное условие** (CompulsoryCondition) – то, без чего невозможно решение задачи. В реализации каркаса этот элемент представлен в виде строковой переменной, формат и логика обработки которой должна предусматриваться создателем задачи. Начальное условие и секрет также представлены в виде строковых переменных и соответственно предполагают создание логики их обработки и интерпретации.

Начальное условие (InitialCondition) – то с чего следует начинать процесс решения.

Ответ – действие соперника по отношению к игроку.

Вопрос – действие игрока по отношению к сопернику. Ответы и вопросы – это элементы диалога, сообщения которыми они обмениваются. В листинге 2 (приводится далее), в методах VerifyQuestionLogic и GetAnswerLogic представлены точки расширения, которые требуют определения логик проверки вопросов и выдачи соответствующего ответа.

В рамках логики проверки вопросов (VerifyQuestionLogic) необходимо выдать **результат обработки вопроса** (QuestionVerificationResult). Результат может быть следующим: корректный вопрос, некорректный вопрос, повторный вопрос.

В рамках логики ответа на вопрос (GetAnswerLogic) необходимо указать состояние решения задачи (GamePuzzleState). **Состояния решения задачи**: почти решена, решена, не решена.

Процесс решения. Состояние процесса решения: активный, неактивный. Если задача решена полностью, логика каркаса переводит процесс в неактивное состояние. **Правила**

(GameRules). Объекты класса следят за исполнением логики проверки вопросов (VerifyQuestionLogic).

Шаг. В рамках процесса решения задачи шаг представляет собой последовательность действий – задать вопрос, получить ответ. **Статистика шагов** (StepsStatistics). Логика каркаса фиксирует каждый шаг в рамках статистики. Статистика учитывает: состояния решения задачи, временные отметки, вопросы.

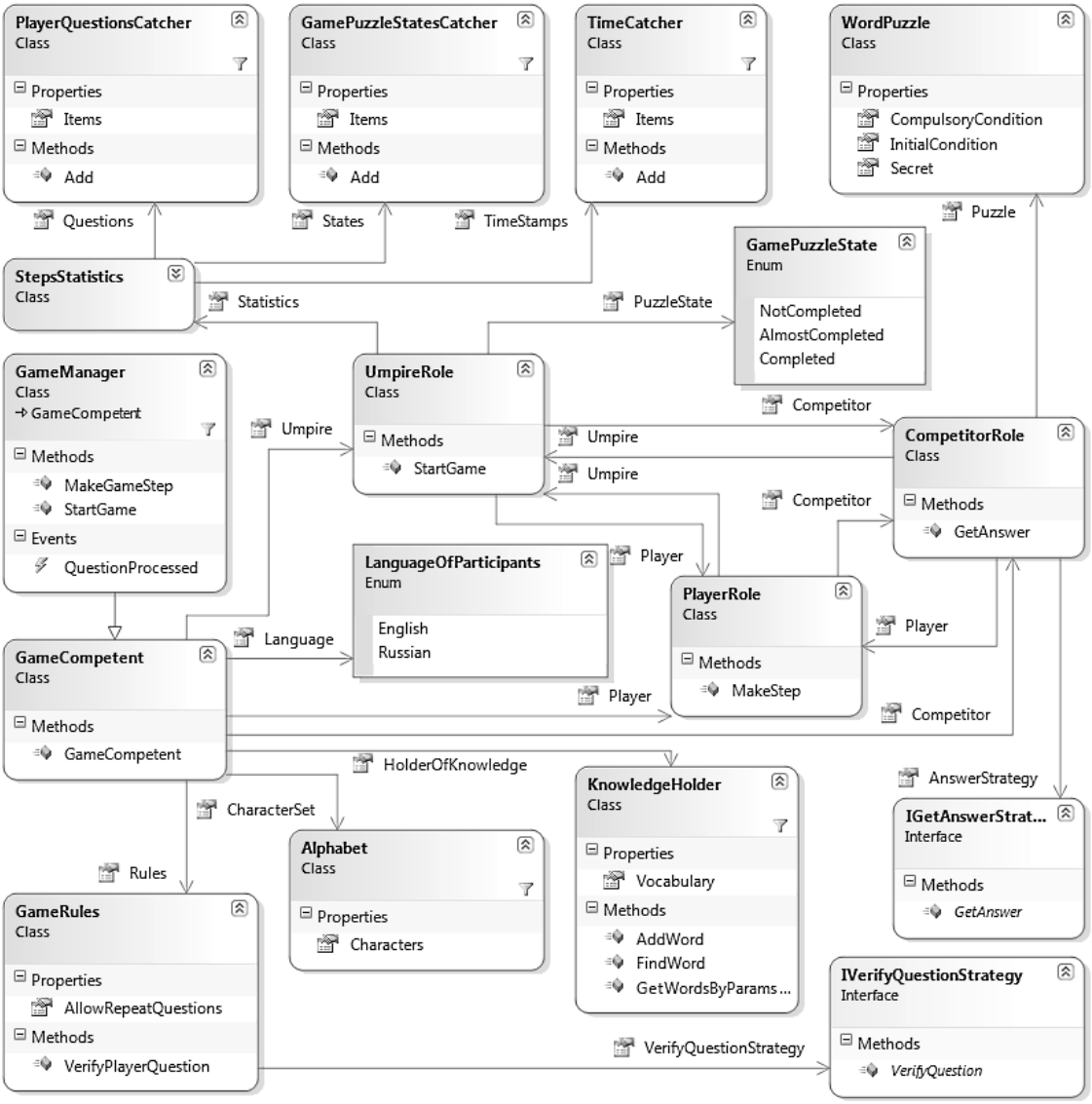


Рис. 1 – Диаграмма классов каркаса

В листинге 1 представлен код шаблона проекта игрового компонента, позволяющий облегчить разработку компонентов с помощью каркаса. В листинге 2 представлен код, в котором этот шаблон реализуется. Цель программного кода, предложенного в листингах – выделить точки расширения повторно-используемой части каркаса.

Листинг 1. Код шаблона проекта

```
using System;
using GSEducation.AQLPGFramework;
namespace GSEducation.AQLPGFramework.DetalizationTemplate{
    //2 step - "manager class" creation by inheritance
    public partial class TemplateGameManager: GameManager{
        //"default constructor" creation
        public TemplateGameManager()
            : base(LanguageOfParticipants.Russian, true){
            //6 step - invoke "game initialization method"
            InitializeGame();}
```

```
            //4 step – "customizable constructor" creation
        public TemplateGameManager(LanguageOfParticipants language, bool
allowRepeat) :
            base(language, allowRepeat){
            //6 step – invoke "game initialization method"
            InitializeGame();}
        //5 step – "game initialization method" implementation
        private void InitializeGame(){
            //14 step – invoke "make puzzle method"
            MakePuzzle();}
        //13 step – overridable "make puzzle method" implementation
        protected virtual void MakePuzzle() {
            throw new NotImplementedException();}}

    //7 step – "get answer strategy class" creation by interface
    public partial class TemplateGetAnswerStrategy : IGetAnswerStrategy{
        //9 step – "customizable constructor" creation + arguments
        public TemplateGetAnswerStrategy(GameManager manager){
            gameManager = manager;}
        private GameManager gameManager;
        //8 step – implement "get answer interface"
        public GamePuzzleState GetAnswer(string[] question, out string answer){
            GamePuzzleState puzzleState = GamePuzzleState.NotCompleted;
            answer = "";
            //step 18 – invoke "get answer logic method"
            puzzleState = GetAnswerLogic(question, out answer);
            return puzzleState;}
        //step 17 – "get answer logic method" implementation
        protected virtual GamePuzzleState GetAnswerLogic(string[] question, out
string answer){
            throw new NotImplementedException();}}

    //10 step – "verify question strategy class" creation by interface
    public partial class TemplateVerifyQuestionStrategy :
IVerifyQuestionStrategy{
        //12 step – "customizable constructor" creation + arguments
        public TemplateVerifyQuestionStrategy(GameManager manager){
            gameManager = manager;}
        private GameManager gameManager;
        //11 step – implement "verify question interface"
        public QuestionVerificationResult VerifyQuestion(string[] question){
            QuestionVerificationResult result =
QuestionVerificationResult.Correct;
            //20 step – invoke "verify question method"
            VerifyQuestionLogic(question, out result);
            return result;}
        //19 step – "verify question logic method" implementation
        protected virtual void VerifyQuestionLogic(string[] question, out
result){
            throw new NotImplementedException();}}
}
```

Листинг 2. Предоставление точек расширения

```
using System;
using GSEducation.AQLPGFramework;
namespace GSEducation.AQLPGFramework.DetalizationTemplate{
    class CustomizedGameManager : TemplateGameManager{
        public CustomizedGameManager(LanguageOfParticipants language, bool
allowRepeat)
            : base(language, allowRepeat){}
        public CustomizedGameManager()
            : base(){ }
        protected override void MakePuzzle(){
            //15 step – "verify question strategy" initialization
                Rules.VerifyQuestionStrategy =
```

```
                new CustomizableVerifyQuestionStrategy(this);
            //16 step - "get answer strategy" initialization
            Competitor.AnswerStrategy = new
                CustomizableGetAnswerStrategy(this);
            //VARIABLE PART - EXTENSION POINT
            //================================
            //================================
        }
        private class CustomizableVerifyQuestionStrategy :
TemplateVerifyQuestionStrategy
        {
            public CustomizableVerifyQuestionStrategy(GameManager manager)
                : base(manager){ }
            protected override void VerifyQuestionLogic(string[] question,
              out QuestionVerificationResult verificationResult){
                verificationResult = QuestionVerificationResult.Correct;
                //VARIABLE PART - EXTENSION POINT
                //================================
                //================================
            }}

        private class CustomizableGetAnswerStrategy : TemplateGetAnswerStrategy{
            public CustomizableGetAnswerStrategy(GameManager manager)
                : base(manager){ }
            protected override GamePuzzleState GetAnswerLogic(string[] question,
                out string answer){
                GamePuzzleState puzzleState = GamePuzzleState.NotCompleted;
                answer = "";
                //VARIABLE PART - EXTENSION POINT
                //================================
                //================================
                return puzzleState;}}}
}
```

Места в листинге 2, обозначенные как «VARIABLE PART», представляют собой точки расширения, в которые разработчик может вставлять изменяемый код (активно пользуясь элементами каркаса). Листинг 1 содержит код шаблона проекта, который абсолютно не требует изменений, он не содержит точек расширений и служит своеобразным проектным решением, способствующим более комфортной работе по созданию конечного компонента на основе каркаса. В листинге 2 для каждой новой разработки следует изменять название класса «CustomizedGameManager» на более подходящее в конкретном контексте имя.

2. Лингвистическая задача «What Letter I Thought»

Описание задачи. Условие – случайным образом загадывается буква, которую необходимо отгадать игроку. Игра состоит из двух этапов. На первом этапе игрока просят последовательно называть слова, в которых предположительно содержится загаданная буква. Если при очередном вопросе игрока, выраженном в виде требуемого слова, обнаружится, что это слово содержит загаданную букву, игрок переводится на следующий этап. На этом этапе он должен задавать вопросы в виде букв. Т.е. первый этап завершается тогда, когда игрок находит слово, в котором есть загаданная буква, теперь на основе этого, игрок должен найти, что же это за буква. Когда буква будет найдена, задача считается решенной.

Логика создания задачи-головоломки (точка расширения метода MakePuzzle() в листинге 2) заключается в следующем. Необходимо выбрать случайным образом букву, которую игрок должен угадать. В данном случае можно воспользоваться такими элементами предметной области, как GameCompetent.CharacterSet.Characters и CompetitorRole.Puzzle.Secret (Листинг 3). Случайным образом выбирается буква из алфавита текущего языка. Её значение становится секретом задачи-головоломки.

Логика стратегии проверки вопроса игрока (точка расширения метода VerifyQuestionLogic(..)) представлена в листинге 3. После того, как игрок сделал ход, задал

вопрос, предложив слово, происходит его проверка в соответствии с текущим этапом игры. Если состояние решения задачи на данный момент – ещё не решена, значит сейчас первый этап, если – почти решена, то второй. На первом этапе вопрос проверяется по словарю, на втором – по алфавиту. Выносится соответствующее решение о корректности или некорректности заданного вопроса.

Логика стратегии выдачи ответа (точка расширения метода GetAnswerLogic(..)) представлена в листинге 3. Суть такова – анализируется вопрос игрока. Если его вопрос, он же – требуемое слово (либо буква, если игрок на втором этапе), содержит секрет задачи, т.е. букву, которую нужно угадать, то в зависимости от того, на каком этапе находится игрок, решается, перевести его на второй этап или же поздравить с победой. На первый взгляд, кажется, что игрок на первом же этапе может назвать требуемую букву, не переходя на второй этап. Однако это невозможно, т.к. вышеописанная логика стратегии проверки вопроса игрока на первом этапе сравнивает его вопрос на принадлежность словарю. А словарь, по крайней мере, входящий в реализацию описываемого каркаса, не содержит слов длиною в одну букву.

Листинг 3. Лингвистическая задача «What Letter I Thought»

MakePuzzle() extension point:

```
Random randomizer = new Random();
List<string> alphabet = CharacterSet.Characters;
if (alphabet != null){
    int currentIndex = randomizer.Next(0, alphabet.Count - 1);
    if (alphabet.Count > currentIndex){
        Competitor.Puzzle = new WordPuzzle();
        Competitor.Puzzle.Secret = alphabet[currentIndex];}}
```

VerifyQuestionLogic(..) extension point:

```
verificationResult = QuestionVerificationResult.Incorrect;
//1st stage
if (gameManager.Umpire.PuzzleState == GamePuzzleState.NotCompleted){
    if (gameManager.HolderOfKnowledge.FindWord(question[0]))
        verificationResult = QuestionVerificationResult.Correct;}
//2nd stage
if (gameManager.Umpire.PuzzleState == GamePuzzleState.AlmostCompleted){
    if (question[0].Length == 1 &&
        gameManager.CharacterSet.Characters.Contains(question[0])){
        verificationResult = QuestionVerificationResult.Correct;}}
```

GetAnswerLogic(..) extension point:

```
if (question[0].Contains(gameManager.Umpire.Competitor.Puzzle.Secret)){
    if (gameManager.Umpire.PuzzleState == GamePuzzleState.NotCompleted)
        puzzleState = GamePuzzleState.AlmostCompleted;
    else if (gameManager.Umpire.PuzzleState == GamePuzzleState.AlmostCompleted)
        puzzleState = GamePuzzleState.Completed;}
else{
     if (gameManager.Umpire.PuzzleState == GamePuzzleState.AlmostCompleted &&
        question[0].Length == 1){
        puzzleState = GamePuzzleState.AlmostCompleted;}
}
```

Выводы

В работе представлен процесс детализации объектно-ориентированного каркаса. В качестве примера приводится реализация алгоритма простой лингвистической задачи в рамках этого каркаса, для демонстрации особенностей программного кодирования. В листингах 1 и 2 даётся проектное решение, позволяющее сделать максимально комфортным процесс детализации каркаса, путём выделения точек расширения.

Реализация каркаса зарегистрирована в отраслевом фонде электронных ресурсов [8].

Литература

1 Приёмы объектно-ориентированного проектирования. Паттерны проектирования / Гамма Э. [и др.]. СПБ.: Питер, 2008. 366 с.

2 Гусс С.В. Элементы повторного использования для программных систем учебного назначения. Концептуальное проектирование и анализ // Математические структуры и моделирование. Омск: ООО «УниПак», 2009. Вып. 20. С. 78-92.
3 Гусс С.В. Использование компьютерных лингвистических игр в процессе обучения // Открытое образование. Москва: «CAPITALPRESS». 2010. № 1. С. 18–29.
4 Гусс С.В. Высокоуровневая модель семейства программных компонентов для поддержки занятий в игровой форме // Математические структуры и моделирование. Омск: «Омское книжное издательство». 2010. № 21. С. 44–54.
5 Гусс С.В. Модель каркаса программных компонентов поддержки занятий лингвистической направленности в игровой форме // Прикладная информатика. Москва: ООО «Маркет ДС Корпорейшн», 2010. Вып. 3 (27). С. 62-77.
6 Гусс С.В. Проблема повторного использования в разработке семейства игровых программных систем учебного назначения // Вестник Омского университета. Омск: Издательство ОмГУ им. Ф.М. Достоевского. 2010. №4. С. 147–149.
7 Гусс С.В. Пакет вспомогательных средств для построения семейства программных систем в определённой предметной области // Программная инженерия. Москва: «Новые технологии». 2011. №1. С. 25–33.
8 Свидетельство о регистрации электронного ресурса ОФЭРНиО Государственной академии наук № 15369. Каркас для создания программных компонентов поддержки занятий лингвистической направленности в игровой форме / Гусс С.В. (Российская Федерация); дата регистрации: 12.02.2010.

УДК 681.513

АВТОМАТИЗИРОВАННАЯ ИНФОРМАЦИОННАЯ СИСТЕМА «ДЕКАНАТ»

Мазорчук Мария Сергеевна, к.т.н., доцент, Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ», Украина, Харьков, mazorchuk_mary@inbox.ru
Тыжненко Юлия Вячеславовна, Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ», Украина, Харьков, vylyashkadidi@yandex.ru

Введение

Одним из наиболее перспективных способов эффективного управления образовательным процессом является внедрение информационно-коммуникативных технологий в процесс менеджмента на базе международных стандартов [1,2].

Развитие сферы информационных технологий дало толчок к усовершенствованию всех сфер жизнедеятельности общества. Достаточно актуальным является вопрос автоматизации учебного процесса, в частности, автоматизации процессов управления в высших учебных заведениях (вузах). Цель автоматизации управления – получить удобные средства мониторинга, анализа и отображения основных процессов приема студентов, административного управления, подготовки и выпуска специалистов университета (института, академии).

Следует отметить, что одной из основных и трудоёмких в работе вуза является деятельность деканата.

Работникам деканатов приходится выполнять огромный объем рутинной работы по учету контингента студентов, обеспечению учебного процесса, предоставлению информации в различные подразделения вуза. При этом всю информацию необходимо представлять в различных форматах. При этом необходимо осуществлять прогноз успеваемости студентов с целью эффективного контроля за процессом обучения и своевременного реагирования на возможные негативные результаты (низкие показатели абсолютной и качественной успеваемости). Таким образом, необходимость внедрения информационной системы, автоматизирующей основные функции образовательного процесса, является ***актуальной***.

Прежде чем внедрять автоматизированную информационную систему (АИС) в деканате, как и в любом другом подразделении вуза, необходимо определить основные

требования к её работе. Выявленные проблемы предметной области помогут определить направление автоматизации данной сферы деятельности, а разработка модели предметной области обеспечит успешную разработку и внедрение базы данных, как основной части АИС.

1. Постановка задачи исследования

Целью данной работы является разработка автоматизированной информационной системы управления документооборотом деканата высшего учебного заведения.

Задачи исследования:

- провести анализ проблемы автоматизации высших учебных заведений, выявить основные задачи, которые необходимо решать в процессе управленческой деятельности деканата;
- разработать концептуальную и физическую модели базы данных хранения информации о студентах;
- обосновать выбор методов и моделей обработки и анализа данных по результатам обучения студентов;
- провести анализ и обосновать выбор существующих программных инструментариев для разработки систем хранения информации и осуществления эффективного документооборота;
- реализовать в виде конечного программного продукта систему «Деканат», которая обеспечит ввод, хранение и анализ информации о студентах и результатах их учебы.

Объектом исследования является процесс ввода, хранения и анализа информации о результатах обучения студентов.

Предметом исследования являются математические методы и модели обработки информации об успеваемости.

Методы исследования: в ходе работы применялись методы системного анализа для описания предметной области и проектирования базы данных; методы проектирования информационных систем для разработки АИС «Деканат»; методы статистического анализа данных для обработки информации об успеваемости студентов. Для разработки конечного программного продукта АИС «Деканат» была применена технология разработки конфигурации фирмы 1С, что позволило реализовать основные функции документооборота в удобном для пользователя виде.

Математическая постановка задачи:

Исходные данные:

$S = \{s_i\}_{i=1..N}$ – множество студентов, обучающихся в вузе;

$Y = \{y_j\}_{j=1..M}$ – множество дисциплин, которые преподаются в вузе;

$P = \{p_{ij}^k\}_{i=1..N, j=1..M, k=1..K}$ – множество оценок студентов, где k – номер семестра.

Основные параметры анализа:

Абсолютная успеваемость A, которая определяется как отношение количества сдавших студентов сессию N' к общему числу N по всему множеству студентов S и всем дисциплинам Y, k-м семестре, по l-й группе, соответственно:

$$A^{SY} = \frac{N'^{SY}}{N^{SY}};\ A^k = \frac{N'^k}{N^k};\ A^l = \frac{N'^l}{N^l}.$$

Качественная успеваемость Q, которая определяется как отношение количества студентов, успешно сдавших сессию (на «отлично» и «хорошо») N'' к общему числу по всему множеству студентов S и всем дисциплинам Y, в k-м семестре, по l-й группе соответственно:

$$Q^{SY} = \frac{N''^{SY}}{N^{SY}};\ Q^{k} = \frac{N''^{k}}{N^{k}};\ Q^{l} = \frac{N''^{l}}{N^{l}}.$$

Статистические параметры анализа успеваемости (средняя успеваемость, мода, медиана, среднеквадратичное отклонение, максимальная и минимальная оценка и т.д.).

Научная новизна полученных результатов состоит в следующем:

Разработана система управления документооборотом в деканате по результатам обучения студентов, которая позволяет избегать повторяющихся операций разработки, дублирования информации, рутинных методов обработки. Разработанный программный продукт совместим с системами 1С.

Практическое значение полученных результатов. Спроектированная база данных и автоматизированная информационная система обеспечит быстрое получение информации, необходимой для своевременного контроля и управления процессом обучения, построения рейтинга студентов и прогнозирования их успеваемости.

2. Реализация информационной системы

Одним из структурных подразделений вуза, наиболее очевидно требующих автоматизации, является деканат.

Деканат – подразделение вуза, осуществляющее контроль и организацию деятельности факультета.

В деканате проводится большой объём рутинной, повторяющейся работы, что, безусловно, обеспечивает неправильное распределение рабочего времени сотрудников деканата, тормозя и запутывая основные процессы деятельности деканата.

Учёт и наблюдение контингента студентов является актуальной задачей в связи с потребностью деканатов в оперативном:

- формировании таких документов, как зачётно-экзаменационные ведомости, ведомости по предметам, сводные ведомости, ведомости по студентам;
- получении любой требуемой информации по конкретным студентам, а также по архивным данным.

Из этих документов извлекается вся необходимая информация, касающаяся учебного процесса.

Исходя из основных назначений деканата, определяем, что основной вид его деятельности – это делопроизводство. Основной функцией деканата является координация и административное обеспечение учебного процесса.

Исследуя предметную область деканатов, можно определить большое число процессов, которые можно условно сгруппировать в несколько пунктов:

- организация и управление учебным процессом;
- заполнение учётных документов;
- контроль и прогнозирование успеваемости студентов (диаграммы успеваемости и рейтинги).

Основными документами, функционирующими в деканате, являются:

- личное дело студента и преподавателя;
- информация о родителях;
- ведомость на каждого студента;
- ведомость по предметам;
- сводная ведомость;
- формирование дипломных приложений;
- формирование формы №10;
- формирование учебного плана;

Все эти документы требуют своевременного заполнения, для обеспечения учётной работы в деканате.

Контроль успеваемости студентов производится в результате экзаменационных сессий, где анализируется успеваемость каждого студента. В результате контроля успеваемости становится возможным полный контроль и прогнозирование результатов будущих сессий.

На данный момент многие вузы не имеют автоматизированной системы управления документооборотом. Большинство документов заполняются вручную несколько раз, информация дублируется, возникают сложности при поиске необходимых данных. Эти проблемы можно устранить при помощи разработки и внедрения АИС [3].

Для разработки концептуальной и физической моделей БД АИС деканата необходимо определить основные требования к системе, выделить сущности и связи между ними, определить перечень документов, на основании которых контролируется учебный процесс.

Наиболее рутинными, но в то же время важными процессами в работе сотрудников деканата являются:

- ввод персональных данных студента в его личные дела;
- ввод персональных данных преподавателя в его личные дела;
- ведение архива деканата, на который прямо или косвенно опирается ряд дополнительных структур института (бухгалтерия, отдел кадров и др.);
- подготовка различных отчетов (учебная карточка студента, экзаменационная ведомость и др.);
- заполнение следующих документов: сводных ведомостей, ведомостей по предметам, ведомостей по студентам, учебные планы для групп, личные дела студентов и преподавателей;
- формирование рейтингов и успеваемости студентов;
- формирование дипломных приложений;
- прогнозирование успеваемости студентов.

Облегчение условий труда достигается благодаря возможности автоматизировать основные процессы ведения делопроизводства. В рамках АИС «Деканат» должны быть реализованы основные функции ввода, сохранения, редактирования, анализа и вывода результатов по состоянию учебного процесса в вузе.

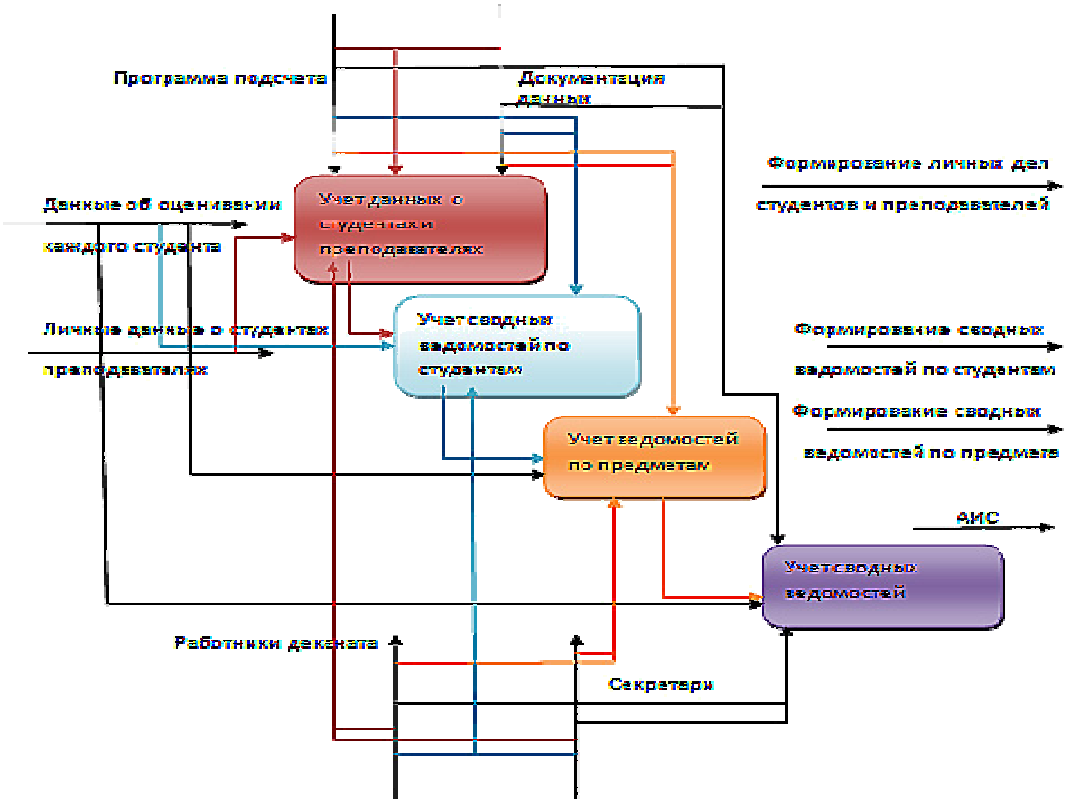


Рис.1 – Визуальное представление модулей в среде BPWin

Деканат участвует в оперативном формировании различных видов документов. Основными из них являются зачетно-экзаменационные ведомости, сводные ведомости, личные дела студентов и преподавателей, ведомости по предметам, ведомости по студентам, отчеты по успеваемости, дипломные приложения, учебный план.

Отчётные документы создаются для получения любой требуемой информации по конкретным студентам, а также по архивным данным.

Сводная ведомость составляется на одного студента и является одной из форм контроля успеваемости студента. Она содержит такие основные данные: ФИО, группа, курс, сессия, семестр, учебный год, название предмета, оценки.

Личное дело составляется на отдельного студента. Оно содержит основные сведения о студенте из его личной карточки: ФИО, группа, факультет, кафедра, место проживания, курс, факультет, семейное положение, форма обучения, куратор, адрес, дата рождения, телефон, идентификационный номер.

Для создания БД необходимо рассмотреть процессы документооборота в деканате. Структура этих процессов представлена на рисунке 1.

Так как все данные, поступающие в деканат (входящая документация), отображаются в различных отчетах – сводной ведомости, ведомостях по предметам, по студентам, то данные могут вводиться в базу данных путем «Ввода на основании», что дает возможность избавиться от дублирования одной и той же информации. Затем можно составить стандартный набор запросов для упрощения выборки данных и на основе составленных запросов сгенерировать отчёты, которые бы выводили необходимую информацию. Естественно, эти отчёты можно распечатать и таким образом получить готовые отчётные документы.

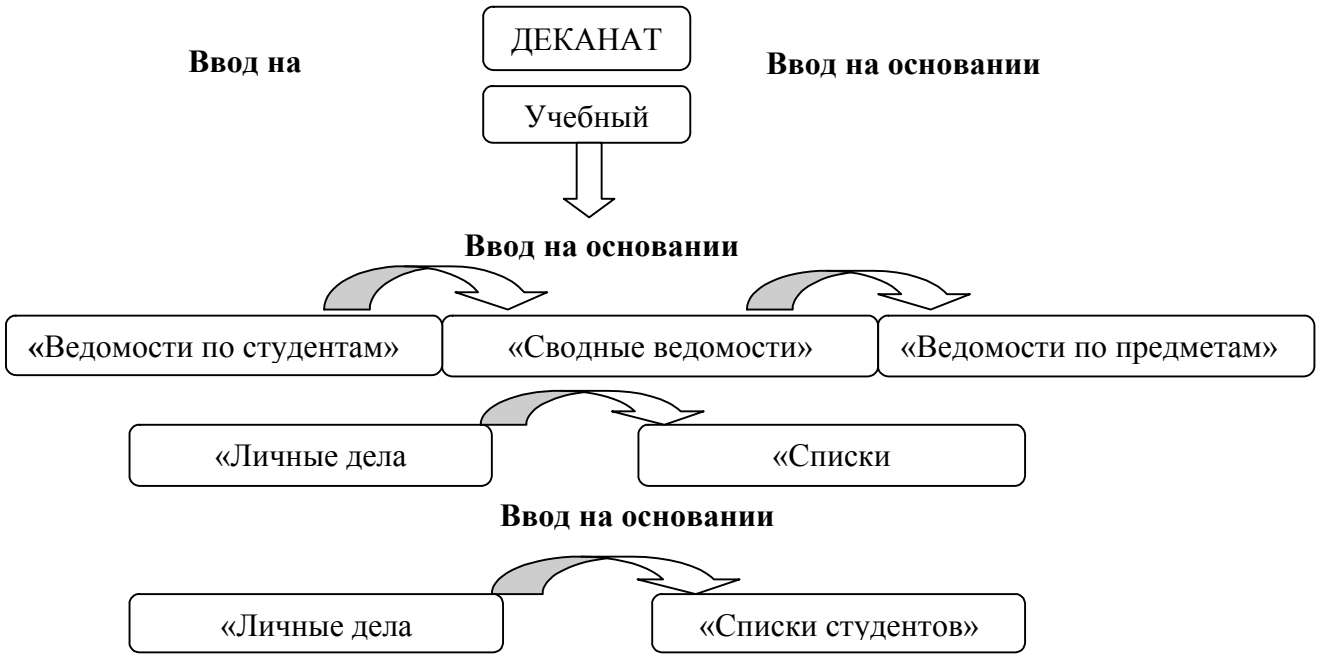


Рис. 2 – Схема формирования документов в системе 1С

На основе структуры процесса документооборота в деканате были выделены основные сущности и атрибуты предметной области, установлены связи между объектами и разработана концептуальная и физическая модели БД.

Для разработки компьютерной системы расчета параметров был выбран программный инструментарий 1С:Предприятие.

При помощи созданного программного продукта были разработаны следующие объекты: личные дела студентов и преподавателей, некоторые виды ведомостей и документов, а также построены рейтинги успеваемости студентов групп 345 и 345-а факультета Системы управления летательными аппаратами (СУЛА) Национального аэрокосмического

университета им. Н.Е. Жуковского «ХАИ» (в виде графиков и диаграмм). Рейтинговые показатели отображают уровень успеваемости и активности студентов [4]. Схема формирования документов в 1С представлена на рисунке 2.

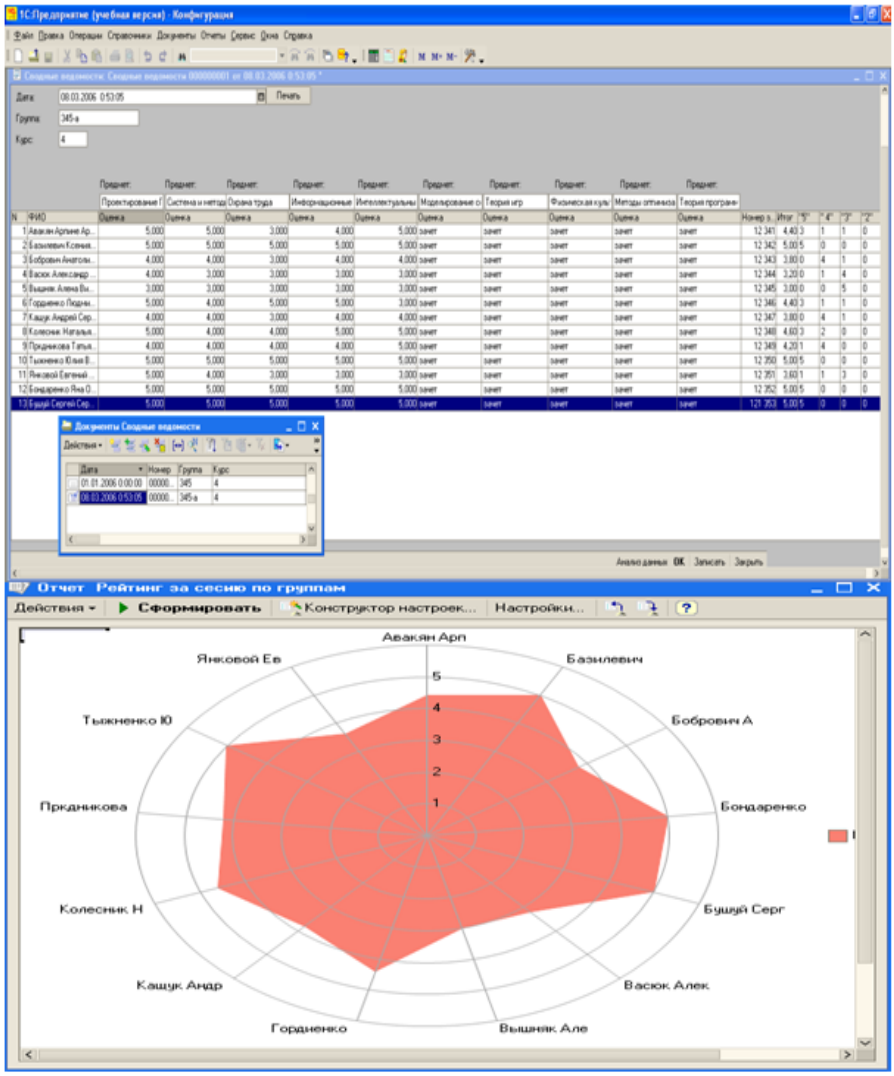


Рис. 3 – Результат работы программы

В системе были созданы следующие виды справочников и документов: «Списки преподавателей», «Личные дела преподавателей», «Списки студентов», «Личные дела студентов» «Факультеты», «Учебный план» «Ведомости по студентам» и «Ведомости по предметам», «Сводные ведомости». Многие данные в этих документах требуют многократного ввода дублирующей информации. Для того чтобы облегчить ввод повторяющихся данных в разных документах, в программе использовалась функция «Ввод на основании», т.е. повторяющиеся информация автоматически дублировалась по всех документах, что облегчило работникам деканата их формирование. На рис. 3 представлены графические формы приложения.

Выводы

Разработанная АИС «Деканат» решает задачи тактического уровня управления. Данная система позволяет повысить эффективность работы деканатов высших учебных заведений.

Для успешного и эффективного функционирования вуза в целом необходимо комплексное внедрение информационно-коммуникационных технологий, охватывающее все без исключения функциональные подсистемы учебного заведения (такие как деканаты, библиотека, приёмная комиссия, бухгалтерия, кафедры и др.)

Литература

1. Фофаев Э.В. Разработка и эксплуатация удалённых БД. – ИЦ Академия, 2008. – 1242с.
2. Гекалов И.А. БД: от проектирования до разработки приложений. – ИЦ Академия, 2003. – 3084с.
3. Головков А.В. Автоматизированная система планирования и контроля учебной деятельности. – [электронный ресурс] – http://conf.bstu.ru/conf/docs/0037/1396.doc.
4. Автоматизированная система управления учебным заведением. – [электронный ресурс] – http://www.mkr.org.ua .

УДК 681.3

РЕАЛИЗАЦИЯ ИНСТРУМЕНТАЛЬНОЙ СРЕДЫ СЕМАНТИЧЕСКОГО МОДЕЛИРОВАНИЯ УЧЕБНОГО ПРОЦЕССА[1]

Грегер Сергей Эдуардович, доцент, Уральский федеральный университет имени первого Президента России Б.Н.Ельцина», Нижнетагильский технологический институт, Россия, Нижний Тагил, segreger@gmail.com

Распространение дистанционного образования и внедрение информационно-коммуникационных технологий в образовательный процесс выявили растущую потребность как в разработке образовательных электронных изданий (ОЭИ), так и в способах электронного представления нормативных документов. Обычно содержимое ОЭИ сильно зависит от учебного плана соответствующей дисциплины и должно быть скорректировано при изменении последнего.

Для специальностей, связанных с одной предметной областью (например, с областью информационных технологий) наблюдается интеграционный характер их дидактических единиц. Так, такая дидактическая единица, как тема дисциплины может иметь множественное представление в разных учебных курсах, иметь различное лексическое представление, но семантически представлять собой одну тему учебной дисциплины. Вместе с этим, одноименные темы учебных курсов могут поддерживаться различными дидактическими единицами – презентациями лекций, описаниями лабораторных работ и т.п., использование которых определяется степенью подготовки обучаемых, целями учебного курса, временными ограничениями учебного плана.

Для разрешения этих противоречий при создании инструментальных средств эффективным является использование семантических моделей предметных областей. В основе семантических моделей лежит понятие сети, образованной помеченными вершинами и дугами. В семантической модели проблемная область представлена в виде сети, представляющей собой набор вершин и связей. В качестве вершин в сети могут выступать классы объектов, объекты и свойства из проблемной области.

В качестве семантических моделей при разработке программного обеспечения чаще всего используются онтологии и словари (таксономии). Онтология – спецификация концептуализации или явное, формальное описание предметной области. Как и в объектно-ориентированном описании, онтология состоит из классов и их экземпляров. У классов и экземпляров выделяются свойства, на свойства могут накладываться логические ограничения.

Представленная инструментальная среда предназначена для построения семантической модели учебного процесса и управления и отображения различных его составляющих. Инструментальная среда реализована как специализированный веб-сервис, входящий в состав портала дистанционного обучения.

При разработке программного продукта ставились цели предоставить пользователю возможность:

[1] Лауреат номинации "**Лучший доклад о методах преподавания объектных технологий в ВУЗе**". Автор доклада награждается правом бесплатной публикации одного доклада по данной тематике на следующей конференции

- создание модели учебного процесса как системы специальностей, учебных дисциплин и обеспечивающих их учебных курсов в соответствии с разработанной онтологией;
- разработки моделей электронных курсов как системы учебных тем и обеспечивающих их учебных объектов;
- создания учебных объектов, как элементов ресурсного обеспечения учебного курса;
- определения соответствия тем курса и соответствующих им ресурсов.

Для обеспечения данных возможностей была создана онтология, включающая в себя соответствующе концепты и связи между ними. При ее построении были использованы два известных подхода к проектированию содержания образования – модели Д.Ш. Матроса [7] и модели, предложенной Т.И. Михеевой и И.Е. Михеенковым [8].

Д.Ш. Матрос предложил автоматизировать проектирование содержания образования на основе структурной целевой модели – совокупности взаимосвязанных целевых единиц, отражающих отдельные элементы содержания обучения [7, с. 41-42]., а их взаимосвязи отображены на ориентированном графе. Т.И. Михеева и И.Е. Михеенков [8] рекомендуют для проектирования содержания гипертекстовых обучающих программ применять таксономическую модель.

Следование этим принципам выражено в использовании двух типов связей – связи «родитель-потомок» и отношения ассоциации. Выразить всю сложность взаимосвязей между отдельными элементами дисциплины или курса, используя только иерархическое отношение, невозможно. Поэтому простая таксономия используется только в отдельных случаях, когда можно четко выделить семантически предопределенную иерархическую структуру, или когда для создания такой структуры вводится дополнительная концептуальная единица. Так при организации учебных объектов можно ввести концепт «Медиа» для группировки всех мультимедийных элементов. Для устранения такого ограничения используется отношение ассоциации между концептами и соответствующими им компонентами. Этот тип отношений позволяет устанавливать связи между элементами, находящихся в разных таксономиях. Использование двух видов отношений позволяет строить над единой базой элементов системы различные семантические сети и использовать их как для целей обучения, так и для целей управления учебным процессом. В нашем случае была разработана онтология, отображающая концептуальную модель предметной области. Онтология отражает семантическую структуру учебной информации. Прикладное приложение призвано поддерживать ее визуализацию в формах, ориентированных на управление учебными материалами и их изучение. Часть концептуальной схемы онтологии, отображенной в редакторе онтологии Protege-OWL, представлена на рисунке (Рисунок 1).

Онтология создается как совокупность двух частей – части, описывающей модель концептов (классов) с указанием их свойств и связей между концептами, и части, представляющую собой базу знаний, состоящую из реализаций классов (объектов). Аналогичным образом прикладную систему можно рассматривать как совокупность двух подсистем, одна из которых представляет собой подсистему управления метаданными, а вторая – подсистема обработки данных.

Существует противоречие между объектной моделью, используемой для представления метаданных, и объектной моделью клиентского приложения. Для хранения метаданных важно наличие унифицированной формалистики описания метаданных, и объектная модель, используемая для отображения этого, будет оптимизирована с точки зрения представления метаданных. Объектная модель прикладного приложения должна быть «заточена» под конкретную предметную область. Таким образом, даже в рамках одной объектной базы должны существовать две системы объектов, одна их которых отображает онтологическую модель в выбранной спецификации, а другая – предметную область.

Обычно для хранения онтологий используются решения, основанные на использовании реляционных баз данных. В нашем случае в качестве слоя хранения была выбрана объектная база данных ZODB [6]. Управление базой и интерфейс пользователя обеспечивается системой управления содержимым Plone.

В данной реализации инструментальной системы не предусмотрена возможность добавления в онтологию новых концептов и связей между ними. Это позволило создать набор компонентов, каждый из которых предназначен для объектно-ориентированного представления соответствующего концепта онтологии.

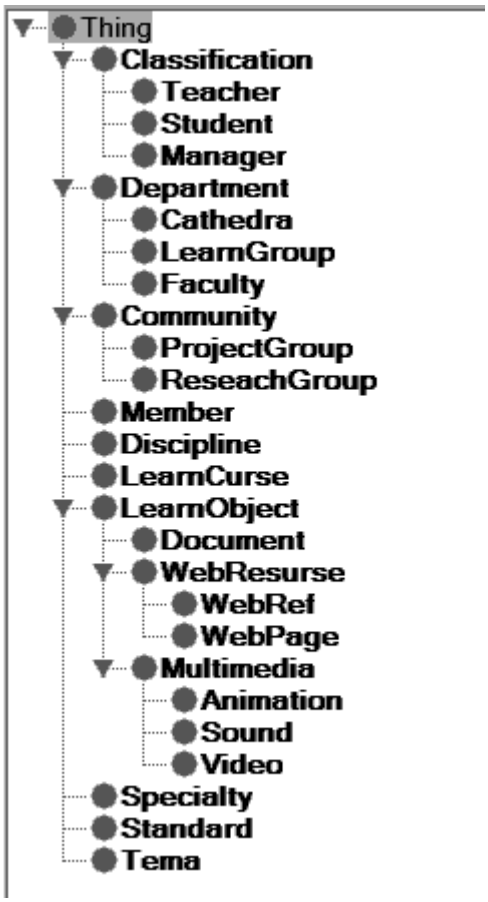


Рис. 1 – Структура онтологии

Такая реализация позволяет пользователю оперировать привычными понятиями «Учебный курс», «Специальность», «Учебная тема», «Учебный объект».[3,4,5], Связи между концептами в онтологии реализованы соответствующими атрибутами классов, хранящих множество ссылок на связанные объекты. Объектная диаграмма UML представлена на рисунке (Рисунок 2).

Каждый компонент инкапсулирует методы, обеспечивающие хранение объектов в объектной базе. Система управления содержимым автоматически генерирует формы редактирования и формы отображения объекта, используя описание атрибутов класса. Кроме этого, каждый компонент обладает методами, позволяющими генерировать представление его реализаций в спецификации OWL. Использование этих методов в рекурсии по объектной базе позволяет получать текстовое представление онтологии в спецификации OWL.

Подобная организация компонентов обеспечивает два уровня представления онтологии. Для функционирования системы используется внутреннее по отношению к ней представление онтологии через набор объектно-ориентированных классов и реализующих их объектов. Для внешнего представления используется представление онтологии в спецификации OWL, что позволяет использовать различные инструментальные средства, производящие семантическую обработку.

С точки зрения упрощения использования системы недостаточно квалифицированным пользователем операции над экземплярами этих компонент, предельно ограничены и сведены к операциям «Добавить», «Удалить», «Копировать», «Вставить», «Установить ссылочную связь». Такой подход позволил создать достаточно простой интерфейс пользователя.

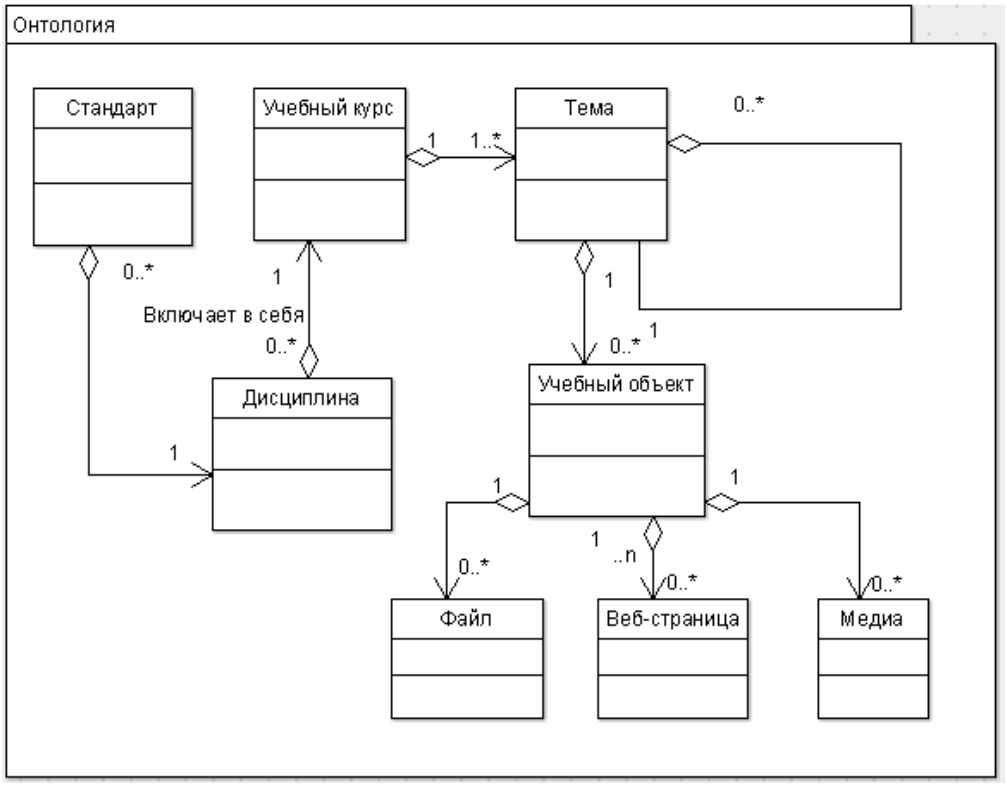


Рис. 2 – Объектная диаграмма

Инструментальная система реализована как веб-приложение [1, 2, 3], и используется в составе сайта кафедры информационных технологий Нижнетагильского технологического института.

Литература

1. Грегер С.Э. Сервер приложений «Zope». Учебное пособие для вузов М.:Горячая линия – Телеком, 2009.-256 с.:ил.
2. Грегер С.Э. Администрирование и интерфейс пользователя CMS Plone (монография). Федер. Агентство по образованию, ГОУ ВПО "УГТУ-УПИ им.первого Президента России Б.Н.Ельцина". Нижнетагил. технол. ин-т (фил.). - Нижний Тагил: НТИ(ф) УГТУ-УПИ, 2009. – 140с.
3. Грегер С.Э. Пакет компонентов обеспечения информационной поддержки образовательного процесса для учебного портала на базе CMS Plone. Тезисы докладов V конференции «Свободное программное обеспечение в высшей школе» . М:Институт логики 2010г. с.15-17
4. Грегер С.Э. Разработка дополнительных компонентов для обеспечения информационной поддержки образовательного процесса для учебного портала на базе CMS Plone. Новые образовательные технологии в вузе: сборник материалов седьмой международной научно-методической конференции, 8 – 10 февраля 2010 года. В 2-х частях. Часть 1. Екатеринбург: ГОУ ВПО «УГТУ-УПИ имени первого Президента России Б.Н.Ельцина», 2010. С.97-100
5. Грегер С.Э. Реализация задач дистанционного обучения средствами CMS Plone. Актуальные вопросы использования инновационных технологий в образовательном процессе: Материал всероссийской научно-практической конференции, Нижний Тагил, Россия, 2010г. /НТГСПА – Нижний Тагил, 2010.С 166-169.
6. Грегер С.Э., Сковородин Е.Ю. Построение онтологического портала с использованием объектной базы // Объектные системы – 2010: Материалы I Международной научно-практической конференции. Россия, Ростов-на-Дону, 10-12 мая 2010 г / под общ. ред. П.П. Олейника. – Ростов-на-Дону, 2010. С. 74-78.
7. Матрос Д.Ш. Электронная модель школьного учебника // Информатика и образование. – 2000. – № 8. – С. 41-42.
8. Михеева Т.И., Михеенков И.Е. Программная таксономия – основа для создания гипермедийных обучающих программ // Информационные технологии. – 1998. – №8 – С. 40-43

УДК 681.3.06

СРЕДА WEB-ПУБЛИКАЦИИ ОБЪЕКТОВ ДЛЯ СТУДЕНТОВ

Неудачин Илья Георгиевич, к. ф.м. н., доцент, Уральский федеральный университет имени первого Президента России Б.Н.Ельцина, Россия, Екатеринбург, nigs@sky.ru

Введение

Описывается учебный курс освоения Zope. Zope – это сервер, среда, инфраструктура разработки и выполнения объектных Web-приложений с открытыми исходными кодами. Этот курс лекций обучает тому, как разрабатывать, отлаживать и сопровождать Web-приложения при помощи Zope. Он заинтересует как пользователей и администраторов Zope, так и тех, кто только начинает изучение Zope.

Технически Zope представляет собой объектно-ориентированную среду, реализующую на основе Python базовую идею заполнения стандартных объектов данными портала и данными, извлеченными и отображенными из запросов пользователя [1]. Любое информационное наполнение в Zope представляется как объекты определенного типа.

1. Описание среды Z Object Publishing Environment (Zope)

Среда Z Object Publishing Environment (Zope) удовлетворяет большинству требований для построения динамического Web-сайта, и в этом убеждают её характеристики. К выбору Zope могут привести две основные причины. Во-первых, Zope – открытый продукт (Open source). Это позволяет студентам, при желании, не только видеть, как что-то работает, но понимать, как это работает. Во-вторых, использование Zope не требует начальных вложений (если не считать времени). Это позволяет изучать, исследовать и приобретать опыт без больших начальных расходов. В-третьих, Zope можно изменять. Мотивы внедрения и изучения: легкость, с какой в Zope можно создавать интересные приложения; активность пользовательского сообщества Zope[2], уже создавшего сотни нетривиальных и полезных компонент Zope; наконец, сама интегрирующая мощь Zope. Zope включила в себя почти все: объектную ориентированность, базы данных, транзакционные системы, публикацию, Web-протоколы, XML, распределенное составление контента, системы управления проектами, E-коммерцию, порталы.

2. План и аннотации лекций

2.1. Установка и старт Zope

Эта лекция объясняет, как установить и впервые запустить Zope. К концу лекции вы должны установить и начать работу в Zope. Довольно легко установить Zope на большинстве платформ, и обычно для этого требуется не больше десяти минут. Изучается структура каталога установки. Настраивается экземпляр Zope и проводится обучение элементарному администрированию [3].

Цель лекции. Изучить процедуры поиска, загрузки, установки и запуска инфраструктуры Zope. Описать локализацию, настройку экземпляра и эксплуатацию Zope. Сообщается, как получить помощь при неполадках. К концу этой лекции вы должны установить Zope и начать в ней работу.

2.2. Концепции архитектуры Zope

Вводятся фундаментальные понятия и частично описывается архитектура Zope. Объектная структура Zope является иерархической, т.е. типичный сайт Zope состоит из объектов, которые содержат другие объекты (которые могут содержать другие объекты, и т.д. до бесконечности). В дополнение к этому, объекты Zope умеют заимствовать динамическое поведение других объектов. Заимствование развивает традиционное наследование классов объектов принципом контейнерного вложения [3]. Объекты Zope хранятся в объектной базе данных ZODB (Zope Object DataBase).

Цель лекции. Объяснить, что такое инфраструктура Zope и для кого она предназначена. В общих чертах описывается, что можно делать с помощью Zope. Сообщить о различиях между Zope и другими серверами Web-приложений. Изучить фундаментальные понятия Zope (объектный подход и заимствование) и описать архитектуру Zope.

2.3. Интерфейс управления Zope

Когда вы загружаете Zope, вы попадаете в интерфейс управления Zope (Zope Management Interface – ZMI). ZMI – среда управления и разработки, которая позволяет вам управлять Zope, объектами Zope, и разрабатывать Web-приложения через Web. Лекция описывает функции и применение ZMI.

Цель лекции. Изучить интерфейс управления Zope (Zope Management Interface – ZMI). Научить пользоваться фреймами ZMI для управления объектами: редактирование, импорт и экспорт. К концу этой лекции студент должен быть в состоянии перемещаться по пространству объектов Zope, копировать и перемещать объекты и использовать другие основные возможности Zope.

2.4. Применение основных объектов Zope

Разрабатывая Web-приложение в Zope, мы конструируем приложение из объектов. В этой лекции описываются три группы основных объектов Zope: объекты контента, объекты презентации и объекты логики. Изучаются фундаментальные объекты (папки, документы, методы, файлы, графические файлы) и действия над ними. Основными операциями над выбранными объектами являются: копировать, вырезать, вставить, удалить, экспортировать, импортировать.

Цель лекции. Описать три группы объектов Zope, предназначенных для: хранения контента, Web-презентации контента, и программирования логики обработки данных. Изучить действия над фундаментальными объектами: папка, документ, метод, файл. Здесь будет построено простое приложение с использованием объектов шаблонов страниц и скриптов.

2.5. Основы скриптинга в Zope

Эта лекция рассказывает, как сделать Web-запрос объекта Zope и как управлять объектами Zope из приложения, используя методы и скрипты Python. Изучается и применяется программный интерфейс API приложения Zope.

Цель лекции. Изучить процедуру Web-запроса как объекта Zope. Обсудить идею скриптов в Zope, и сосредоточиться на скриптах Python. Освоить создание и применение скриптов на основе Python и программного интерфейса API приложения Zope. Научить методике вызова других объектов из скриптов.

Объяснить, как программировать деловую логику в Zope с помощью более мощных инструментов, нежели DTML. Показать, как добавить производительную силу скриптинга в Web-сайт.

2.6. Основы шаблонов страниц Zope

Шаблоны страниц Zope (Zope Page Templates – ZPT) – это инструмент генерации Web-страниц и средство быстрого скриптинга графического интерфейса основанное на XML. Они помогают программистам и дизайнерам сотрудничать при создании динамических Web-страниц для приложений Zope [4].

Цель лекции. Реализовать принципы Web-дизайна и организовать сотрудничество программистов и дизайнеров через шаблоны страниц Zope. Объяснить, как создавать и редактировать шаблоны страниц. Ввести основные операторы шаблонов, которые позволят вставлять динамический контент.

Узнать, как внедрить логику принятия решений в свои шаблоны посредством переменных и условных тестов. Показать, как вся эта теория может быть практически использована, вводом простого процессора формы для применения на Web-странице.

2.7. Циклические структуры шаблонов страниц

После обзора переменных и условных операторов лекция расширяет знание ZPT (шаблонов страниц) изучением циклов, динамически генерированных атрибутов и управлением исключениями.

ZPT включает несколько конструкций циклов, которые позволяют сканировать по последовательностям переменных и объектов. ZPT имеет некоторые довольно интересные отличия от традиционного подхода.

Цель лекции. Дать интенсивный курс циклических атрибутов TAL – Template Attribute Language. Показать, как может использоваться атрибут repeat, чтобы сканировать последовательности. Добавить функциональные возможности стандартных конструкций циклов, существующих в других языках. Демонстрировалось использование реального, динамического построения Web-страницы из базы данных MySQL и одного циклического шаблона.

Построить макет страницы библиотеки файлов в качестве иллюстрации того, как ZPT позволяет динамически определять атрибуты Web-страницы, и также объединять эту возможность с атрибутом repeat для итерационного построения Web-страницы. Наконец, дать беглый обзор простейших возможностей обработки ошибок, доступных в ZPT, как жизнеспособной и очень простой альтернативы встроенному обработчику ошибок Zope.

2.8. Разработка приложений в Zope

В этой лекции построим простые, но мощные Web-приложения при помощи изученных инструментов. Способы проектирования, обсуждаемые здесь, применимы также в сложных случаях.

Лекция приведёт нас шаг за шагом к построению первого Web-приложения в Zope. Дополнительно исследуем некоторые из центральных объектов Zope.

Цель лекции. Привести пользователю несколько практических примеров построения Web-приложений Zope. Объяснить, как использовать основные объекты Zope и как они могут взаимодействовать при формировании приложения. Zope имеет еще много дополнительных свойств, кроме уже рассмотренных, но эти простые примеры должны помочь начать создавать динамические, сложные, но хорошо управляемые Web-сайты.

2.9. Основы динамического контента в DTML

DTML (Document Template Markup Language) – это презентация на основе тегов и скриптовый язык [5]. Это означает, что теги DTML, вложенные в HTML, заменяют части страницы "вычисленным" контентом. Команды DTML выполнятся сервером.

Цель лекции. Познакомить с DTML, скриптовым языком разметки на основе тегов, применяемым в Zope. Описать применение DTML для создания шаблонов страниц и для скриптинга и его место по отношению к другим способам скриптинга в Zope. Объяснить синтаксис DTML и ввести три основных тега: *var*, *if* и *in*. Изучив эту лекцию, можно создавать динамические Web-страницы.

2.10. Web-формы запроса к серверу

Для того чтобы пользователь мог сформировать и отправить запрос серверу через HTML документ, применяются так называемые формы. Это блок документа HTML, заключенный в тегах <form></form>, содержащий различные текстовые поля и кнопки отправления. Формы ввода в Zope взаимодействуют с DTML и с шаблонами страниц.

Цель лекции. Изучить содержание и атрибуты тега form в HTML. Научить методике применения скриптов DTML и шаблонов страниц обработки форм ввода Zope. Реализовать примеры оптимальных форм.

2.11. Пользователи и безопасность

Zope – это многопользовательская система. Однако, вместо того, чтобы полагаться на учётные записи пользователей, обеспеченные операционной системой, под которой всё это выполняется, Zope поддерживает одну или больше её собственных баз данных пользователей. В этой лекции мы подробно рассмотрим управление пользователями, формирование ролей, отображение ролей на права, и организацию режима безопасности сайта Zope.

Цель лекции. Описать, как Zope работает с пользователями, устанавливает их подлинность (аутентификация), разрешает доступ (авторизация), и обсудить другие вопросы, связанные с безопасностью. Например, формирование типовых ролей членов портала и влияние ролей на права. Функции безопасности являются центральными в архитектуре Zope

и должны быть центральными в архитектуре тех Web-приложений, которые вы создаете в Zope.

2.12. Поиск и каталогизация контента

Zcatalog – это встроенный в Zope инструмент поиска. Он позволяет распределять по категориям и искать в Zope объекты всех типов. Можно также использовать его, чтобы находить внешние данные, типа реляционных данных, файлов, и удаленных Web-страниц. В дополнение к поиску можно использовать ZCatalog, для того чтобы организовать коллекции объектов.

Цель лекции. Показать, как осуществлять индексацию и поиск объектов при помощи встроенного поискового средства Zope – объекта каталога ZCatalog. Ввести понятие индексации и обсудить различные примеры внесения в указатель и поиска. Наконец, рассмотреть результаты поиска и метаданные.

2.12. Присоединение реляционной базы данных

При использовании реляционных данных в Zope вы сохраняете все преимущества Zope, включая безопасность, динамическую презентацию и сетевую организацию. Вы можете использовать Zope, чтобы динамически построить доступ к данным, презентацию данных и управление данными.

Цель лекции. Описать, как Zope связывается с внешними реляционными базами данных. Объяснить подход, позволяющий интерпретировать реляционные данные в качестве объектов Zope. Применить методы запросов Z SQL в сочетании с DTML. Затронуть вопросы безопасности и производительности.

Литература

1. Спикльмайр С. и др. Zope. Разработка Web-приложений и управление контентом: Пер. с англ. – М.: ДМК Пресс, 2003. – 464 с.
2. Zope Community. [Электронный ресурс] http://www.zope.org/
3. Amos Latteier, Michel Pelletier, Chris McDonough, ... The Zope2 Book. – Zope Developers Community, 2009. [Электронный ресурс]
3. http://docs.zope.org/zope2/zope2book/
4. Harish Kamath. ZPT Basics. – Melonfire, 2002 [Электронный ресурс] http://www.devshed.com/c/a/Zope/
5. Harish Kamath. DTML Basics. – Melonfire, 2002. [Электронный ресурс] www.devshed.com/c/a/Zope/DTML-Basics-part-1/

УДК 004.78

УНИФИЦИРОВАННАЯ МОДЕЛЬ ДЛЯ ТЕСТИРОВАНИЯ ИНСТРУМЕНТОВ ОБЪЕКТНО-РЕЛЯЦИОННОГО ОТОБРАЖЕНИЯ

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

Шаблоны проектирования – это многократно используемые решения широко распространённых проблем, возникающих при разработке программного обеспечения. По мере приобретения опыта программисты осознают сходство новых проблем с решаемыми ранее. С накоплением ещё большего опыта приходит осознание того, что решение похожих проблем представляет собой повторяющиеся шаблоны. Зная эти шаблоны, опытные программисты распознают ситуацию их применения и сразу используют готовое решение, не тратя время на предварительный анализ проблемы [1-2].

В настоящее время применение шаблонов (паттернов) проектирования непосредственно связано с их реализацией в объектно-ориентированном языке программирования при помощи объявления иерархии классов и описания ассоциаций. Стоит отметить, что часть паттернов посвящены принципам реализации объектной системы на основе реляционной системы и носит название методов (шаблонов) объектно-реляционного отображения (ОРО) [3]. Т.к. в классической реляционной модели отсутствует

понятие наследования (например, наследование отношений), то реализация этой ключевой концепции объектно-ориентированного языка программирования в виде таблиц базы данных является основным назначением группы методов ОРО, позволяющих представить иерархию классов. В настоящее время существует большое количество программных продуктов, позволяющих прозрачно реализовать объектную систему на основе реляционной базы данных, которые получили название инструментов объектно-реляционного отображения [4-5]. Каждая система имеет собственные принципы преобразования классов в набор отношений и большое количество дополнительных сервисов. Несмотря на большую популярность описанных продуктов, отсутствуют единые правила, по которым можно было бы единообразно сравнить, протестировать их функционал и определить те шаблоны отображения, которые реализованы в них. В данной статье представлена единая объектная модель гипотетического приложения, которая может быть использована для тестирования инструментов ОРО.

Рассмотрим основные методы объектно-реляционного отображения, используемые для представления иерархии классов в виде таблиц реляционной базы данных (рис. 1) [3-4]. Метод Наследование с одной таблицей (Single Table Inheritance) физически представляет иерархию наследования классов в виде одной таблицы реляционной базы данных, столбцы которой соответствуют атрибутам всех классов, входящих в иерархию. Описываемый способ позволяет отобразить структуру наследования и минимизировать (а иногда обойтись и вовсе без соединений) количество соединений (естественных внутренних или открытых), которые необходимо выполнить для извлечения информации. В данном методе каждому экземпляру класса соответствует одна строка таблицы. При создании объекта значения заносятся только в те столбцы таблицы, которые соответствуют атрибутам класса, а все остальные остаются пустыми (имеют null-значения, null-маркеры).

Выполняя загрузку объекта из таблицы в память, необходимо знать имя класса, представляемого в языке программирования. Для этого к таблице добавляется специальное поле. Это может быть имя класса или какое-нибудь идентификационное поле. Перед загрузкой данных необходимо считать код типа класса, чтобы узнать, экземпляр какого производного класса должен быть создан для загрузки объекта. При сохранении объекта в базе данных запись кода типа класса выполняется суперклассом.

Описанный метод Наследование с одной таблицей является одним из вариантов отображения иерархии наследования классов на таблицы реляционной базы данных и имеет следующие ключевые достоинства:

- В структуре БД присутствует лишь одна таблица, представляющая все классы иерархии.
- Для извлечения экземпляров классов иерархии не нужно выполнять соединение таблиц.
- Перемещение полей из базового класса в производный (так же из производного в базовый) не требует внесения изменений в структуру таблицы.

Несмотря на наличие множества достоинств, описанному методу присущи серьёзные недостатки:

- При изучении структуры таблиц БД могут возникнуть проблемы, ведь не все имеющиеся столбцы таблицы предназначены для описания каждого класса предметной области. Это усложняет процесс доработки системы в будущем.
- При наличии глубокой иерархии наследования с большим количеством атрибутов многие столбцы могут иметь пустые значения (null-маркеры). Это приводит к неэффективному использованию свободного пространства в БД. Однако современные СУБД способны сжимать строки, содержащие большое количество отсутствующих значений.
- Итоговая таблица может оказаться слишком большой и содержать огромное число столбцов. Основной способ оптимизировать запрос (сократить время выполнения) – создать покрывающий индекс. Однако множество индексов и большое количество запросов к одной таблице способно привести к частым блокировкам, что будет оказывать негативное влияние на производительность программного приложения.

Альтернативным решением для описанного является метод Наследование с таблицами для каждого класса (Class Table Inheritance), представляющий иерархию наследования классов,

использующий по одной таблице для любого класса. Атрибуты класса отображаются непосредственно на столбцы соответствующей таблицы. При использовании данного метода ключевой является задача соединения соответствующих строк нескольких таблиц БД, представляющих единый объект предметной области. Частым решением выступает применение единого значения первичного ключа во всех таблицах. Поскольку таблица суперкласса содержит по одной строке на каждую строку каждой таблицы производных классов, первичные ключи записей должны быть уникальными в пределах всех таблиц производных классов.

Еще одна важная проблема метода – извлечение данных из множества связанных таблиц. Выполнение по одному запросу к каждой отдельной таблице не является эффективным, поэтому чаще всего выполняется естественное соединение нескольких таблиц по равенству значений первичного ключа. При этом заранее неизвестно, к каким именно таблицам следует применить соединение. Некоторые таблицы будут содержать пустые значения, поэтому для выполнения эффективных соединений с такими таблицами применяется внешнее соединение (правое или левое), которое достаточно медленное.

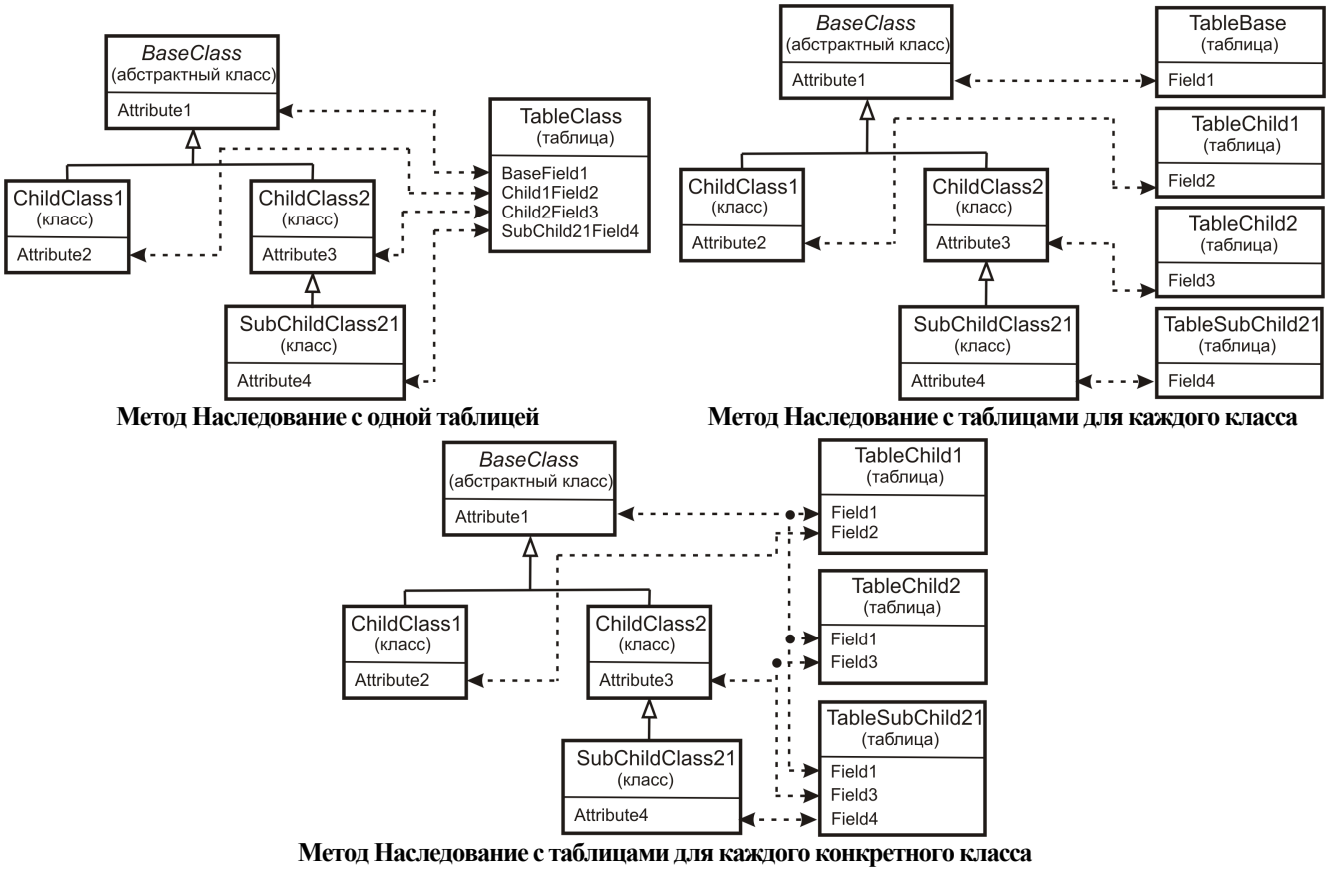


Рис. 1 – Методы объектно-реляционного отображения, используемые для представления иерархии классов в виде таблиц реляционной базы данных

Наследование с таблицами для каждого класса имеет ряд преимуществ:

- В каждой таблице присутствуют лишь поля, соответствующие атрибутам определённого класса. Поэтому таблицы легки в понимании и занимают мало места на жёстком диске.
- Взаимосвязь между объектной моделью и схемой базы данных проста и понятна.
- Однако описанному методу присущи следующие недостатки:
- При создании экземпляра определённого класса предполагается загрузка данных из нескольких таблиц, что требует их соединения либо множества обращений к базе данных с последующим соединением результатов в оперативной памяти.
- Перемещение полей в производный класс или суперкласс требует изменения структуры нескольких таблиц.

- Таблицы суперклассов могут стать слабым местом в вопросах производительности, поскольку доступ к таким таблицам будет осуществляться слишком часто, что приведёт к множеству блокировок.
- Высокая степень нормализации может стать препятствием к выполнению незапланированных заранее запросов.

Метод Наследование с таблицами для каждого конкретного класса (Concrete Table Inheritance) представляет иерархию наследования классов, используя по одной таблице для каждого реализованного (неабстрактного) класса этой иерархии. С практической точки зрения данный метод предполагает, что каждый экземпляр класса (объект), который находится в оперативной памяти, будет отображён на отдельную строку таблицы. При этом каждая таблица содержит столбцы, соответствующие атрибутом как конкретного класса, так всех его предков. Т.е. атрибуты суперкласса дублируются во всех таблицах, представляющих его производные классы.

Преимуществами данного метода является то, что:

- Каждая таблица не содержит лишних полей, вследствие чего ее удобно использовать в других приложениях, в которых не использован инструмент объектно-реляционного отображения.
- При создании объектов определённого класса в памяти приложения и извлечения данных из реляционной базы данных выборка выполнятся из одной таблицы, т.е. не требуется выполнять реляционные соединения.
- Доступ к таблице осуществляется только в случае доступа к конкретному классу, что позволяет снизить количество блокировок, накладываемых на таблицу, и распределить нагрузку на систему.
- Однако описываемому методу отображения присущи следующие недостатки:
- Первичные ключи могут быть неудобны в обработке.
- Отсутствует возможность моделировать отношения (связи) между абстрактными классами.
- Если атрибуты классов перемещаются между суперклассами и производными классами, требуется изменять структуру нескольких таблиц. Эти изменения будут не так часты, как в случае наследования с таблицами для каждого класса, однако их нельзя игнорировать (в отличие от метода Наследование с одной таблицей, в котором эти изменения вообще отсутствовали).
- Если в суперклассе изменить определение хотя бы одного атрибута (например, поменять тип данных), это потребует изменить структуру каждой таблицы, представляющей производный класс, поскольку поля суперкласса дублируются во всех таблицах его производных классов.
- При реализации метода поиска данных в абстрактном классе необходимо просматривать все таблицы, представляющие экземпляры производных классов. Это требует большого количества обращений к базе данных.

После того, как описаны основные методы объектно-реляционного отображения, реализуемые в современных инструментах, рассмотрим унифицированную объектную модель, используемую для тестирования (рис. 2).

Данная иерархия классов позволяет проверить возможности реализации трёх описанных ранее методов объектно-реляционного отображения: 1) Наследование с одной таблицей; 2) Наследование с таблицами для каждого класса; 3) Наследование с таблицами для каждого конкретного класса. Несмотря на то, что для представления должности в гипотетической предметной области выделено три класса, корневым из которых является абстрактный класс Post, а два унаследованных от него – реализованные классы ExperiencePost и ScientificPost соответственно, физически атрибуты всех классов будут представлены в виде столбцов одной-единственной таблицы реляционной базы данных.

Для каждого класса (как абстрактного, так и реализованного), представляющего контрагента, (производного от Contragent) должна быть выделена отдельная таблица, которая будет содержать столбцы для каждого атрибута, объявленного только в данном классе (а также поля, необходимые для организации ассоциаций между классами).

Для представления адресов в предметной области выделено три класса: абстрактный класс Address и два реализованных CompanyAddress и EmployeeAddress. При реализации данной иерархии в структуре таблиц базы данных предполагается создать две таблицы (т.к. у нас два реализованных

класса), каждая из которых будет содержать столбцы, представляющие атрибуты как абстрактного, так и реализованного классов (а также поля для ассоциаций между классами).

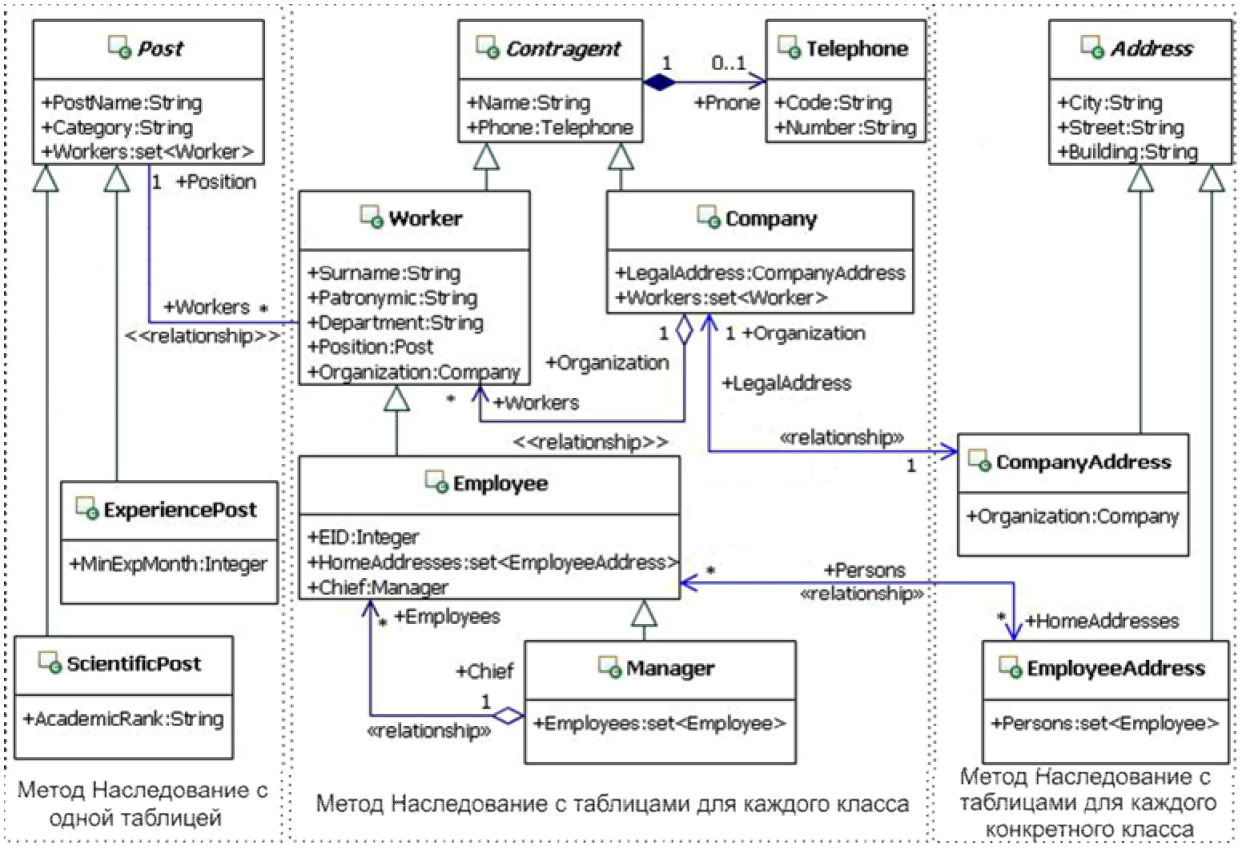


Рис. 2 -Унифицированная модель для тестирования инструментов объектно-реляционного отображения

В данной статье рассмотрена объектная модель иерархии классов, которая может быть использована для тестирования инструментов объектно-реляционного отображения. Дальнейшим развитием данной работы является реализации этой модели с помощью определённого программного продукта, такого как Hibernate [4] или DevExpress XAF [5].

Литература

1. Гамма Э. и др. Приёмы объектно-ориентированного проектирования. Паттёрны проектирования, СПб: Питер, 2001. – 368 с.: ил. (Серия «Библиотека программиста»).
2. Гранд М. Шаблоны проектирования в Java. Пер. с англ. С. Беликовой. – М.: Новое знание, 2004. – 559с.: ил.
3. Флауер М. Архитектура корпоративных программных приложений, Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 544 с.: ил. – Парал. тит. англ.
4. Bauer C., King G., Hibernate in Action, Manning Publications, 2005, 431p.
5. The fastest way to platform independent business applications, http://www.devexpress.com/Products/NET/Application_Framework/

УДК 004.652.4

РАЗРАБОТКА СИСТЕМЫ ПОДДЕРЖКИ ПРИНЯТИЯ РЕШЕНИЙ ПРИ ВЫБОРЕ МУЗЫКАЛЬНЫХ КОМПОЗИЦИЙ ДЛЯ ТОРГОВЫХ ПОМЕЩЕНИЙ

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

Ковтун Нелли Игоревна, старший преподаватель, Старооскольский технологический институт, Россия, Старый Оскол, kovtun-n-i@yandex.ru

Лазарева Татьяна Ивановна, старший преподаватель, Старооскольский технологический институт, Россия, Старый Оскол, tatylaz@mail.ru

Радиовещание давно стало атрибутом жизни современного человека. Миллионы слушателей в разных уголках планеты ежедневно приобщаются к мировой культуре, узнают последние новости, просто отдыхают, включив свои приемники.

Из всех средств массовой информации радио является самым удобным и комфортным для восприятия аудиторией.

Небольшим городским радиокомпаниям приходится конкурировать с сильными соперниками, занимающие высокие рейтинги прослушивания по всей России.

Анализ внутренней и внешней среды радиокомпании показывает, что в рыночное окружение организации входят как действующие конкуренты, так и новые, ещё не зарекомендовавшие себя. Также рыночное окружение включает в себя Российское авторское общество – некоммерческая общественная организация, созданная авторами для реализации и охраны авторских прав в сфере интеллектуальной деятельности.

Британский психолог Дж. Миллиман провел исследование для Британской ассоциации супермаркетов. Оказалось, что музыкальное сопровождение привлекает и направляет внимание посетителей, а также воздействует на скорость их движения [1].

Исходя из потребностей бизнеса радиокомпании, для привлечения новых клиентов и выделения на фоне конкурентов предлагается организовать предоставление дополнительной услуги по подбору музыкальных композиций для торговых и иных помещений.

Для удобства заказчиков предлагается создать представительство в Интернете в виде интерактивного web-сайта, а также систему поддержки принятия решений в выборе музыкальных композиций на основе теории множеств.

Для того чтобы оказывать рассматриваемую услугу, необходимо все имеющиеся данные поместить в единую базу данных, которая будет содержать информацию о клиентах, фирмах (организаций) для которых осуществляется услуга, составителей плейлистов и информацию о музыкальных композициях. Для удобства и наглядности данные необходимо будет поместить в таблицы, т.е. работа будет осуществляться с реляционными базами данных.

Требования к функциональным характеристикам ИС:

1. проектируемая система должна в режиме online давать возможность пользователям сформировать предварительный плейлист;
2. сайт должен содержать форму для ввода информации о помещении, целевой аудитории, загруженности помещения в разное время суток;
3. система должна обеспечивать ведение нормативно-справочной базы данных (БД);
4. обеспечивать формирование выходных форм и отчетов по поставленной задаче;
5. обеспечивать пользователя подсказками по каждому разделу системы.

Требования к надежности:

1. отказы отдельных элементов системы поддержки принятия решений при составлении плейлистов для торговых помещений не должны приводить к ее полному выходу из строя;
2. в системе должны быть предусмотрены средства для прогноза, фиксации и локализации различных нештатных ситуаций и отказов оборудования (таких как: повреждений и перегрузок каналов связи; нарушения целостности БД; попыток несанкционированного доступа в систему и т.д.);
3. должно быть предусмотрено ограничение прав доступа, то есть у всех сотрудников должны быть собственные права доступа к системе, а именно: уникальный пароль и имя входа (они известны только их обладателю и программистам, другим сотрудникам они неизвестны);
4. в системе должны использоваться сложные пароли, состоящие из цифр и букв латинского алфавита разного регистра;
5. так как проектируемая система будет располагаться на сервере, то он должен предусматривать защиту данных от несанкционированного доступа, а также содержать систему шифрования данных.

Условия эксплуатации ИС:

- в конце рабочего дня администратор сайта должен создавать резервные копии баз данных, так как вся информация, необходимая для работы, будет хранится именно там.

Требования к эргономике ИС:

1. экранная форма системы должна быть оформлена в спокойных цветах;
2. все экранные формы выдержаны в одной гамме цветов и одной стилистике;
3. все надписи легко читаемы.

Требования к структуре и функционированию системы ИС:

1. интерфейс системы должен быть лёгким и понятным, даже неподготовленному пользователю;
2. иметь разные уровни доступа в клиентской БД у пользователей;
3. обеспечивать настройку параметров;
4. устойчивость к сбоям связи;
5. легко адаптироваться к постоянно меняющимся параметрам [2].

Выделим входную и выходную информацию решаемой задачи на основе анализа потоков данных в нотации Гейна – Сарсона (рис. 1).

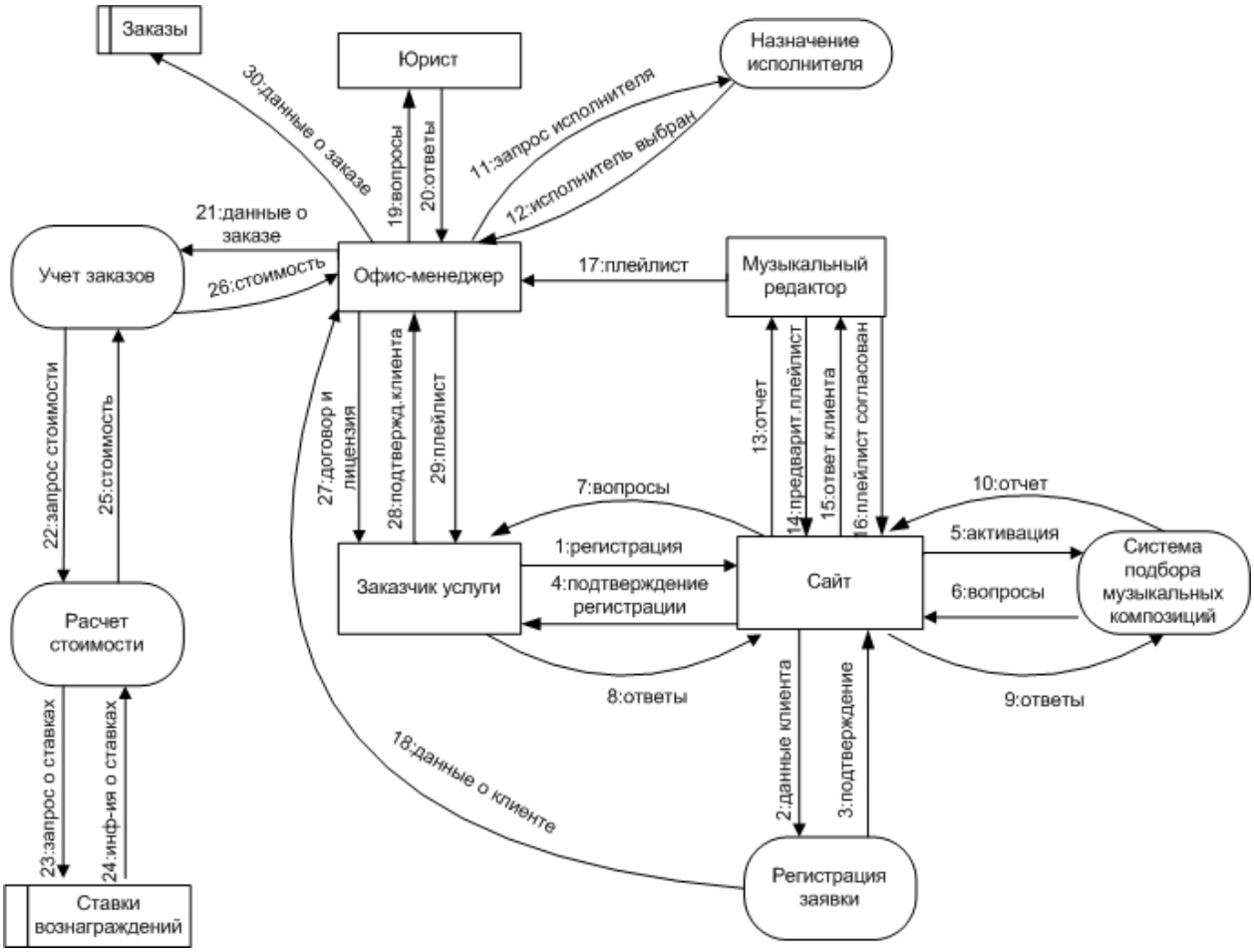


Рис. 1 – Диаграмма потоков данных ИС

В результате анализа диаграммы потоков данных можно выделить входные и выходные данные.

Входной является информация о целевой аудитории, помещении, для которого оставляется плейлист, его продукции, информация о клиенте (владельце помещения), особые пожелания клиента.

Выходной информацией является список подходящих композиций, предварительный и готовый плейлист, договор с клиентом.

На основе анализа предметной области информационно-логическая модель базы

данных может быть представлена в виде совокупности взаимосвязанных сущностей.

Центральной сущностью в инфологической модели является сущность «Плейлист» (рис. 2), который идентифицируется номером плейлиста. Здесь будет храниться информация обо всех плейлистах для рассматриваемой услуги. Сущность характеризуется: номером целевой аудитории, номером клиента, для которого составляется плейлист, номером исполнителя плейлиста, датой составления, также содержит информацию о стоимости заказа, озвучиваемой площади торгового помещения, количестве композиций. Сущность «Плейлист» связана со следующими сущностями:

- «Целевая_аудитория» отношением M:1 «Составляется»;
- «Договор» отношением M:1 «Содержит»;
- «Исполнитель» отношением M:1 «Создает».

Также сущность «Плейлист» связана с сущностью «Музыкальные композиции» отношением 1:M «Содержит», но данную связь можно отразить лишь на инфологическом уровне, т.е. при реализации данные сущности связаны не будут.

Плейлист представляет собой некоторую совокупность музыкальных композиций, хранящихся на сервере, но в самой сущности «Плейлист» не хранятся музыкальные композиции, а указывается информация о количестве песен, целевой аудитории и т.д.

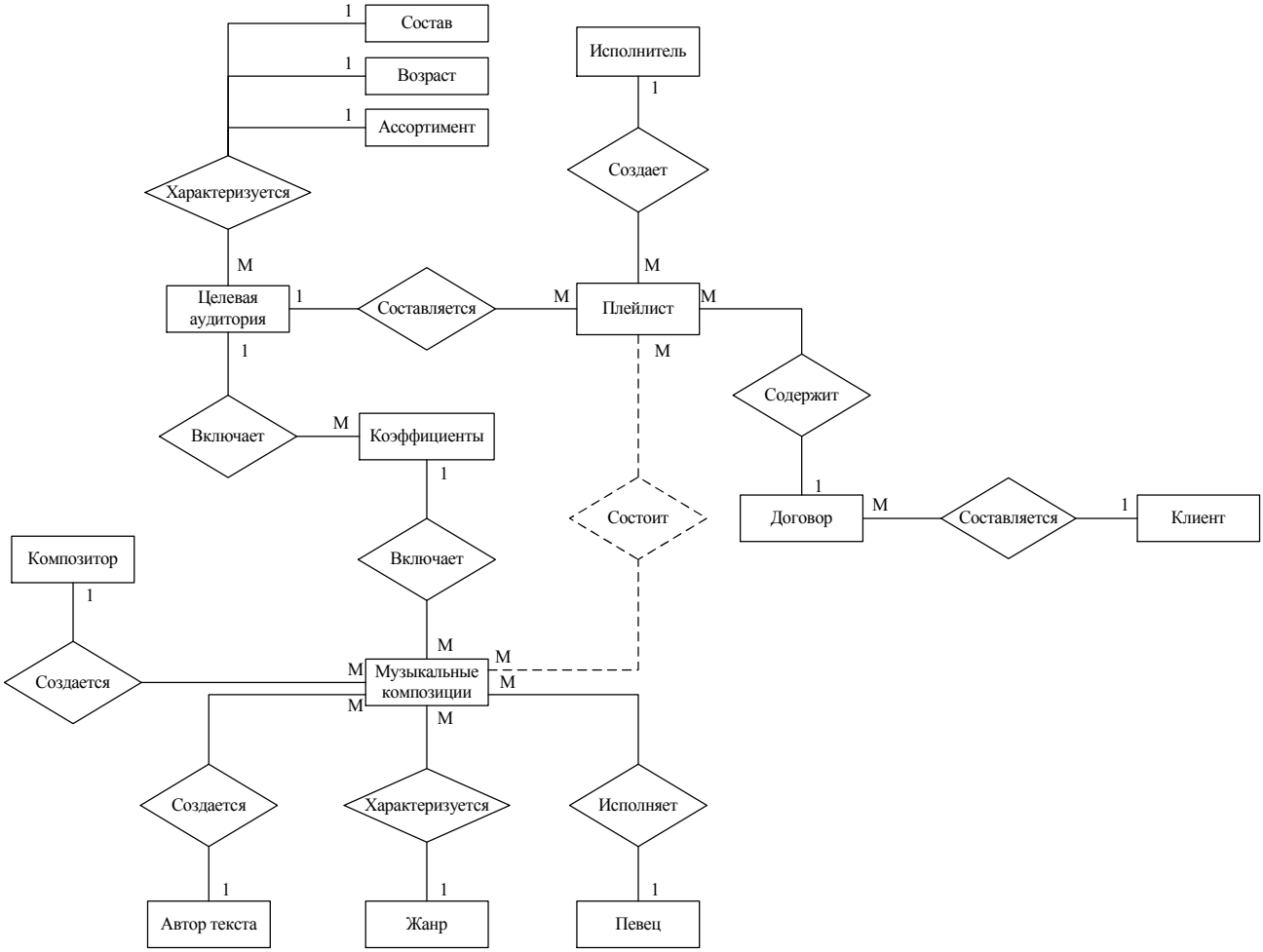


Рис. 2 – Инфологическая модель подсистемы поддержки принятия решений при выборе музыкальных композиций

Разрабатываемая информационная система будет состоять из следующих компонентов: интерфейс пользователя, подсистема поддержки принятия решений, база данных (база знаний), система обработки информации (рис. 3). Пользователь через web-сайт определяет свою целевую аудиторию, на основании которой и осуществляется поиск музыкальных композиций.

Общий алгоритм предоставления услуги по подбору музыкальных композиций имеет следующую последовательность действий: клиент должен авторизоваться на сайте, затем идет проверка правильности ввода данных, после этого клиент переходит на страницу, где расположена форма для построения списка музыкальных композиций. На странице клиенту предлагаются вопросы о его целевой аудитории и вида товара в магазине, для которого формируется плейлист.

После обработки ответов система выводит на экран плейлист с композициями, если клиента он устраивает, то отчет высылается офис-менеджеру радиостанции «Парнас», который назначает музыкального редактора для дальнейшей обработки плейлиста. Клиенту же необходимо заполнить регистрационную форму для ввода более точной информации о фирме, при помощи которой музыкальный редактор сможет более точно подобрать музыкальные композиции. Затем клиенту на почтовый адрес отсылается предварительный плейлист, и если его все устраивает, он посылает положительный ответ. На стороне офис-менеджера происходит расчет цены плейлиста, оформление договора с РАО. Клиенту назначается дата и время, когда он сможет прийти в офис и оплатить услугу, затем ему выдается готовый плейлист и лицензионное соглашение с РАО.

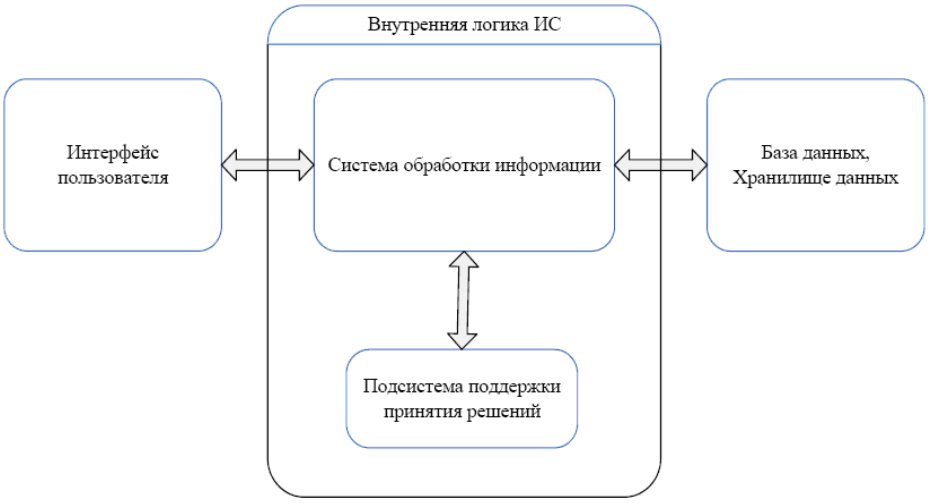


Рис. 3 – Схема взаимодействия подсистемы поддержки принятия решений с компонентами ИС

Таким образом, разрабатываемая информационная система позволит подобрать музыкальные композиции в соответствии с целевой аудиторией и использовать их с соблюдением авторских прав, а также разместить предварительный плейлист в сети Интернет, что позволит получить дополнительную прибыль и увеличить конкурентоспособность фирмы.

Литература

1. Milliman R. E. Using background music to affect the behavior of supermarket shoppers //Journal of Marcetting // <http://en.academic.ru>.
2. Кузнецов С.Д. СУБД (системы управления базами данных) и файловые системы.- М: Майор, 2001 г.

УДК 004.438, 004.08, 004.274

ОА-АРХИТЕКТУРА – НОВЫЙ ПОДХОД К СОЗДАНИЮ ОБЪЕКТНЫХ СИСТЕМ[1]

Салибекян Сергей Михайлович, ст. преподаватель, Московский институт электроники и математики (технический университет), Россия, Москва, salibek@yandex.ru
Панфилов Пётр Борисович, к.т.н., доцент, Московский институт электроники и математики (технический университет), Россия, Москва, panfilov@miem.edu.ru

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

В настоящей статье предлагается новый подход к построению объектных систем – объектно-атрибутная (ОА) архитектура вычислительной системы (ВС), разработанная автором [1,2]. Данный подход обладает намного б**о**льшими возможностями для объектного программирования и создания интеллектуальных систем, чем общепринятое в настоящее время объектно-ориентированное программирование (ООП) и реализует принцип управления вычислительным процессом с помощью потока данных (dataflow).

Объектно-атрибутная (ОА) архитектура основывается на ряде понятий:

Информационная пара (ИП) (атрибутированные данные) – совокупность нагрузки (данных или ссылки на данные), и ярлыка/атрибута (уникального идентификатора), описывающего нагрузку.

Функциональное устройство (ФУ) – это виртуальное (реализованное программным способом) или реальное устройство, осуществляющее обработку данных. ФУ имеет внутренние регистры (набор регистров будем называть контекстом ФУ) и может исполнять некий набор милликоманд, описываемый алгоритмом функционирования этого устройства.

Милликоманда (мК) – это ИП, где ярлык указывает устройству (ФУ), каким образом следует обрабатывать прикрепленные к нему данные.

Капсула – это множество информационных пар, служащих для описания определенного объекта (с помощью капсулы и обеспечивается абстракция данных). Каждая ИП, входящая в капсулу, задаёт один из критериев описываемого объекта.

Например: *Параллепипед {Длина=10 Высота=15 Ширина=20},*

где *{}* – данные знаки задают начало и конец описания капсулы; = – знак сопоставления атрибута и информационной нагрузки; *Параллепипед* – имя информационной капсулы.

Кроме того, в качестве нагрузки в ИП может выступать не только константа, но и ссылка на переменную, капсулу или иную информационную структуру, благодаря чему появляется возможность синтезировать смысловой граф, состоящий из множества капсул, связанных ссылками. Смысловой граф служит для описания сколь угодно сложного объекта подобно тому, как это делается в объектно-ориентированном программировании (ООП) (рис. 1). Капсула может содержать не только данные, но и программу (миллипрограмму), которая состоит из последовательности милликоманд, помещенных в капсулу. Данный подход обладает намного большей гибкостью, чем фреймовая модель, т.к. механизм капсул и ссылок в качестве нагрузок ИП чем-то напоминает детский конструктор, из которого можно собрать какую угодно информационную конструкцию в реальном времени.

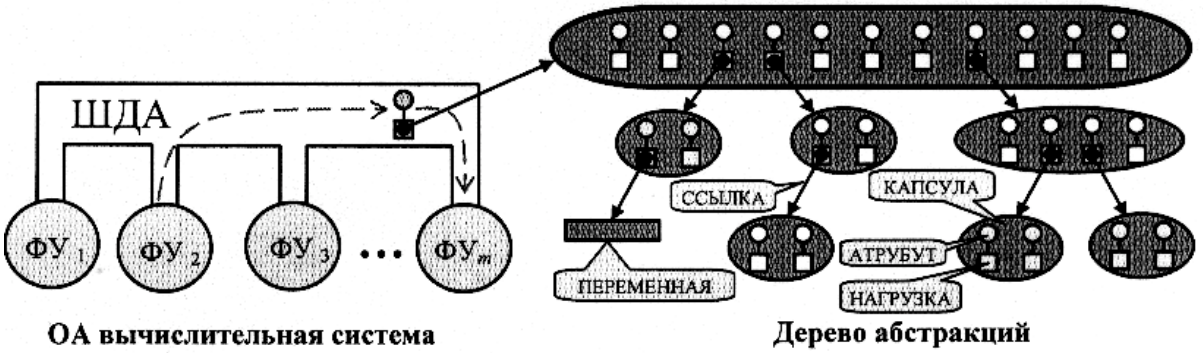


Рис. 1 – Работа ВС с ОА-деревом абстракций

ОА-Дерево абстракций (или смысловой граф) несколько напоминает собой структуру класса в объектно-ориентированном программировании (ООП), однако здесь существует одно весьма важное отличие. Как известно, ООП было строено на основе фреймовой концепции (концепция была разработана Марвином Минским в 70-е годы прошлого века [3]): фреймовая модель была лишь дополнена возможностью ссылки не только на другой фрейм, но и на процедуру или функцию (методы класса объектов). Совокупность фреймов, связанных между собой ссылками, хоть и дает возможность описывать любые реальные или абстрактные объекты, однако не обладает достаточной гибкостью, чтобы изменять свою

структуру непосредственно во время выполнения вычислительного процесса: как правило, фреймовая структура (в ООП классы объектов) создаются ещё до начала выполнения программы и далее не меняются. В ОА-архитектуре используется совершенно другой подход: информационные структуры, описывающие объекты, синтезируются уже во время работы ВС по заранее заложенным программистом (или экспертом) правилам: информационные структуры, образуются путем группировки ИП в капсулы и настройки ссылок между капсулами (синтез дерева абстракций – см. рис. 1).

Синтезом ОА-графа и его анализом занимаются ФУ, использующие новую (милликомандную) командную систему (или ОА-язык). Обмен информацией между ФУ осуществляется через шину данных-атрибута (ШДА), по ней передается милликоманда с нагрузкой (константой или ссылкой на иную информационную конструкцию) (рис. 1). По атрибуту милликоманды ФУ идентифицирует данные, пришедшие к нему по ШДА, и, исходя из атрибута, принимает решение, каким образом ему следует их обрабатывать. Каждое ФУ имеет набор внутренних регистров (контекст), который используется для хранения промежуточных данных.

Например, ФУ *АЛУ* (арифметико-логическое устройство) имеет следующий контекст: аккумулятор, флаг нулевого результата и флаг знака результата. Для выполнения операции сложения на *Шину* необходимо выдать следующую последовательность милликоманд, описанных на ОА-языке:

Bus {ALU.Set=2, ALU.Add=2, ALU.Put=x},

где *Bus* – указание на то, что последовательность милликоманд, расположенных в капсуле, должна быть выдана на Шину; = – знак сопоставления ярлыка и нагрузки; *ALU* – имя ФУ; *ALU.Set, ALU.Add, ALU.Put* – расширенные милликоманды для устройства АЛУ (расширенная милликоманда – это милликоманда, в которой указывается ФУ, что должно милликоманду исполнять: перед точкой стоит обозначение ФУ, которому передается милликоманда, после точки – ярлык передаваемой нагрузки): *ALU.Set* – записать число из нагрузки милликоманды в аккумулятор, *ALU.Add* – сложить значение из аккумулятора с числом из нагрузки и записать результат вычисления обратно в аккумулятор, *ALU.Put* – записать значение из аккумулятора в ячейку памяти, хранящуюся в нагрузке милликоманды; *x* – ячейка памяти, куда АЛУ выдает результат своих вычислений.

ФУ могут быть реализованы как программным, так и аппаратным способом, поэтому ОА-язык одновременно является и языком высокого, и языком низкого уровня.

Например, при программной реализации ФУ алгоритм работы АЛУ можно оформить с помощью следующей процедуры на языке С:

```
void ALU(void * Context, int millicomand, void *Obj),
{
 switch millicomand case
  {
 case 0: // Сброс (Сброс)
         // - в скобках указана мнемоника милликоманды
   PAluContext(Context)-> Accumulator=0;
 case 1: // Установить значение в аккумуляторе (Уст)
   if(Obj==nil) PAluContext(Context)-> Accumulator=0;
   else PAluContext(Context)-> Accumulator=PVariant(Obj);
 case 2: // Сложение (Сум)
   if(Obj<>nil)
    {
     PAluContext(Context)->Accumulator=PAluContext(Context)->
Accumulator+PVariant(Obj);
     if(PAluContext(Context)-> Accumulator>=PVariant(Obj))
       FLager=true;
     else FLager=false;
     if(PAluContext(Context)-> PVariant(Obj)==0)
       FZero=true;
     else FZero=false;
    }
```

```
.....................................................................................
case 4: // Выдать содержимое аккумулятора (Выд)
  if(Obj<>nil)
    PVariant(Obj)=PAluContext(Context)->Accumulator;
   }
   // и т.д.
.....................................................................................
}
```

где *Context* – ссылка на контекст ФУ, т.е. на структуру данных, которая содержит виртуальные регистры (таким образом с помощью одной процедуры реализации логики работы ФУ можно управлять сразу несколькими ФУ лишь меняя адрес в параметре *Context*);

millicomand – индекс милликоманды (по данному индексу ФУ определяет тип данных и их назначение;

Obj – ссылка на нагрузку милликоманды (данная ссылка может указывать как на переменную, так и на любую информационную конструкцию). Ввиду того, что логика работы ФУ довольно проста, не будет особого труда реализовать их не только в программном исполнении (виртуальные ФУ), но и в аппаратном.

Данный интерфейс, состоящий из трёх полей, одинаков для всех типов ФУ, что обеспечивает полную универсальность вызова процедуры реализации логики работы ФУ (в отличие от классических процедур и функций, которые имеют свой индивидуальный интерфейс – перечисление передаваемых и возвращаемых параметров).

Для полноценного же функционирования ОА-системы необходимо ввести ФУ нескольких типов: *Шина* – специализированное ФУ, которое обеспечивает передачу ИП между различными ФУ, *АЛУ*, *ДиспетчерКапсул* – создание, удаление и модификация ИП, группировка ИП в капсулы, *УстройствоВводаВывода*, *ДиспетчерСписков* – осуществляет формирование списков и поиск по списку, исходя из заданных условий, *МатричноеАЛУ* и *Автомат* т.д. Например, ФУ *Автомат* предназначен для того, чтобы выдавать на ШДА последовательность милликоманд, расположенных в капсуле. На рис. 2 представлен фрагмент капсулы с миллипрограммой, которая задает вывод результата вычисления ФУ *АЛУ* на консоль (экран).

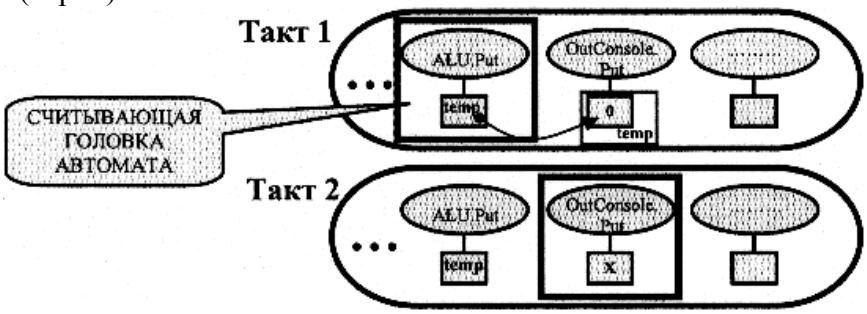


Рис. 2 – Работа Автомата в ОА-системе

На первом такте считывающая головка *Автомата* (указатель на адрес текущей выдаваемой на *ШДА* милликоманды) находится на милликоманде *ALU.Put=temp* («записать результат вычислений в ячейку памяти с адресом temp»). Причем temp – это адрес нагрузки следующей милликоманды); *OutConsole.Put=temp(0)*, где temp задает ссылку на нагрузку, (0) – указывает начальное значение, хранимое в нагрузке. На первом такте *Автомат* считывает первую милликоманду и выдает ее на *Шину*; *Шина* передает милликоманду на устройство *ALU*; а *ALU* записывает результат вычисления в ячейку памяти, адрес которой записан в нагрузке милликоманды, т.е. в temp (нагрузку следующей милликоманды). На втором такте работы *Автомата* указатель (считывающая головка) передвигается на следующую милликоманду и далее вторая милликоманда выдается *Автоматом* на *Шину*; *Шина* передает ее на ФУ *OutConsole*, которое считывает из нагрузки значение, записанное на предыдущем такте *ALU*, и выводит его на консоль. Следует отметить, что в нагрузке может храниться адрес не только переменной, но и любой информационной конструкции (массива, графа абстракций и т.д.). Тип данных, на который ссылается указатель, ФУ определяет по ярлыку:

например, ярлык может быть «Первое слагаемое целого типа», «Первое слагаемое – число с плавающей точкой» и т.п.

ОА-арихитектура в полном объеме обеспечивает принцип управления вычислительным процессом с помощью потока данных: в ОА-системе даже можно обойтись без привычного для классической парадигмы программирования оператора условного перехода (if). Как несложно заметить, ФУ управляются не командами извне, а именно потоком данных, ведь милликоманда по сути является атрибутированными данными, а решение о том, каким образом данные обрабатывать, принимает само ФУ (ФУ выполняет операцию в том случае, когда к нему приходят все необходимые для выполнения операции данные). К тому же, различия между данными и программным кодом стирается благодаря тому, что в ОА-архитектуре полностью совпадают форматы данных и программы: милликоманда – это, по сути, та же ИП, а миллипрограмма – это капсула с набором ИП. Поэтому капсулы с миллипрограммой можно без труда встраивать в ОА-граф, превращая его в мощную базу знаний. В ООП же в основном применяется управление с помощью команд (методы, входящие в классы объектов, представляют собой последовательность команд), что существенно ограничивает гибкость ВС.

К преимуществам разработанной архитектуры можно отнести удобный способ абстракции данных и программного кода, используемый не только для описания, но и для распознания объектов: абстракции не создаются заранее перед выполнением программы, как в ООП, а синтезируются по заранее описанным правилам уже в процессе выполнения программы, что существенно повышает гибкость вычислительного процесса; это позволит наделить ОА-систему полноценным интеллектом: в результате появляется возможность описания/распознания заранее неизвестных программисту объектов и самообучения ВС (т.е. самостоятельное изменение системой правил синтеза информационных конструкций). Синтез же информационных конструкций в ОА-архитектуре осуществляется от простого к сложному: ОА-система распадается на несколько уровней абстракции. Процесс начинается с так называемых информационных атомов – элементарных (неразложимых) абстракций (атом в большинстве случаев представляет собой единственную ИП). Так, для систем автоматизации атомами будут являться сигналы, считанные с датчиков и снабженные ярлыками, идентифицирующими их. Для систем распознавания изображений атомы – описания отдельных пикселов изображения (координаты и код цвета). Для систем смыслового анализа текста атомами будут являться отдельные символы, составляющие анализируемый текст (рис. 3). Например, при смысловом анализе текста на первом уровне абстракции происходит лексический анализ, т.е. формирование слов из символов.

Формированием информационных конструкций для передачи на более высокий уровень абстракции занимаются ФУ, которые реализуют алгоритм первого уровня абстракций. Далее выделенные слова, оформленные в виде информационной капсулы или ОА-графа передаются на следующий уровень абстракции, где происходит синтаксический анализ. Информационные конструкции, сформированные на этом уровне, передаются на уровень выше для смыслового анализа и т.д. На каждом последующем уровне информационные конструкции (абстракции) становятся все более и более сложными и их количество уменьшается (сложная абстракция представляет собой довольно большой смысловой граф). На вершине этого так называемого конуса абстракций находится ключевая абстракция, описывающая распознаваемый объект полностью. Если же необходимо из общей абстракции произвести синтез более простых информационных конструкций, то применяется обратный конус абстракций (т.е. синтез от сложного к простому). Так, если требуется перевести текст с одного языка на другой, то используются сразу и прямой (распознание смысла исходного языка), и обратный (для синтеза текста уже на другом языке) конусы абстракций (рис. 3). ОА-конус абстракций весьма легко реализуется на распределенной ВС: части конуса абстракций относительно независимы друг от друга и их можно расположить на различных вычислительных узлах (рис. 3). К тому же, в отличие от распространенных интерфейсов Corba и DCOM, в ОА-системе нет необходимости описывать

шаблоны интерфейсов удаленных процедур (язык IDL), т.к. все параметры для ФУ передаются по отдельности в виде милликоманд, что значительно облегчает процедуру обмена информацией между вычислительными узлами.

Разработанная архитектура в полной мере поддерживает принцип изоморфизма (безболезненного изменения программы), т.к. любую классическую процедуру возможно реализовать с помощью ФУ: параметры такой процедуры передаются посредством милликоманд. ФУ для других участников информационного процесса представляет собой «черный ящик», общаться с которым можно с помощью милликоманд, поэтому модификация ФУ заключается в расширении набора милликоманд (для совмещения со старыми версиями программ алгоритм работы ФУ следует модернизировать таким образом, чтобы не изменять функционал милликоманд старых), потому изменение количества операндов и последовательность их передачи для ФУ не будут критически влиять на всю ОА-программу. В классических же процедурах и функциях параметры передаются единовременно, и при изменении интерфейса (перечня параметров) процедуры или функции программисту приходится исправлять всю программу. Во-вторых, любое ФУ очень легко заменить с помощью так называемого *Интерпретатора*. *Интерпретатор* представляет собой заглушку, которая переводит последовательность милликоманд для старого типа ФУ в последовательность милликоманд для одного или нескольких новых ФУ. Изоморфизм обеспечивается и на уровне структур данных: модификация ОА-графа, заключающаяся в добавлении в информационные капсулы новых ИП, что не влияет на алгоритмы поиска по графу для других ФУ. Изоморфизм, который обеспечивает ОА-архитектура, не только помогает облегчить труд программиста, но и является необходимым качеством для создания интеллектуальных систем, ведь в процессе распознания объектов программе приходится довольно существенно изменять базу знаний и модификации эти не должны нарушать целостности ВС.

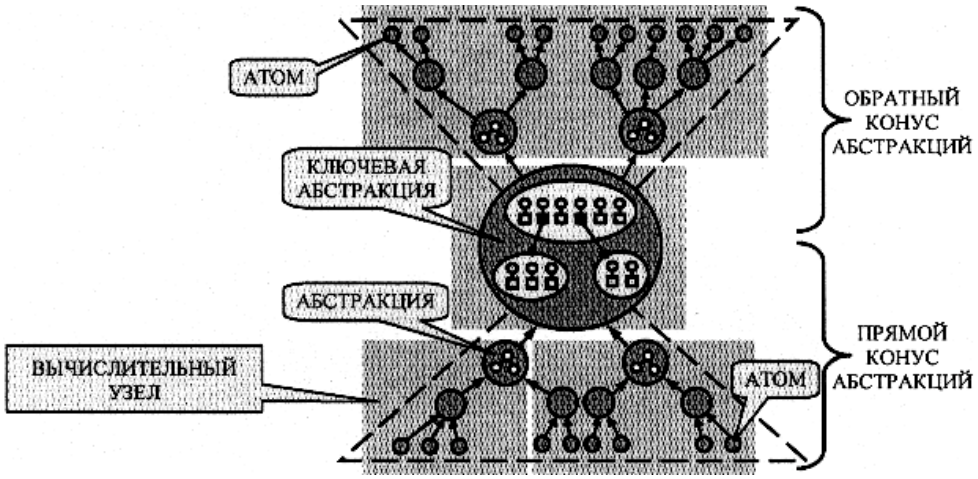


Рис. 3 – Прямой и обратный конусы абстракций

Работоспособность предложенной архитектуры подтверждается тем, что в настоящий момент создана программная платформа ОА-среды. В среде реализованы 35 классов ФУ, ОА-язык программирования (компилятор ОА-языка реализован на базе ОА-архитектуры); имеется возможность не только программирования, но и управления ОА-системой в реальном времени, то есть коррекция алгоритма работы системы без перезагрузки ОА-программы.

В заключение можно отметить следующие положительные стороны ОА-архитектуры:

- удобная структуризация (абстракция) как данных, так и программы;
- изоморфизм;
- обладает «врождённым» параллелизмом вычислений (каждое ФУ работает асинхронно и потому может работать параллельно с другими ФУ);
- легко реализуется на распределенной вычислительной системе (программист не разрабатывает изолированный код для каждого вычислительного узла системы и

затем описывает обмен данными между узлами; а проектирует целостную программу, которая потом будет выполняться на всей распределенной ВС [2]);
- ОА-система реализуется как программно, так и аппаратно;
- при программной реализации обладает легкой переносимостью с одной аппаратной платформы на другую (для того, чтобы ОА-программа заработала на новой аппаратной платформе, необходимо лишь закодировать для новой машины подпрограммы реализации логики работы ФУ);
- обладает хорошей масштабируемостью как на программном, так и на аппаратном уровнях;
- облегчает процесс написания программы: ОА-программа хорошо декомпозируется, части программы весьма легко стыкуются между собой.

Литература

1. Салибекян С.М. Принципы милликомандной архитектуры как основа построения высокопроизводительных адаптивных вычислительных систем // *Автоматизация и современные технологии.* 2002. № 5.
2. Салибекян С.М., Панфилов П.Б. ОА-архитектура построения и моделирования распределенных систем автоматизации // *Автоматизация в промышленности*. 2010 №11
3. Минский М. Фреймы для представления знаний: Пер. с англ.- М.: Энергия, 1979.

УДК 004.4

ОПЫТ РЕАЛИЗАЦИИ УНИФИЦИРОВАННОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ УЧЕТА ОКАЗАННЫХ УСЛУГ[1]

Герасимова Ольга Игоревна, Шахтинский институт (филиал) Южно-Российского государственного технического университета (Новочеркасского политехнического института), Россия, Шахты, itblack@rambler.ru

Введение

Актуальность данной работы обуславливается тем, что в настоящее время все большее количество организации переходит от ведения учета на бумажных носителях к использованию информационных систем (ИС). Корректно спроектированная и реализованная ИС позволяет сделать процесс учета более быстрым и удобным, одновременно способствуя повышению конкурентоспособности предприятия на рынке.

Проектирование структуры базы данных (БД) – одна из наиболее ответственных фаз реализации программного обеспечения, успех которой во многом зависит от выбранной методологии. В данной работе использована методология сущность-связь, подробно описанная в [1-2]. Этап проектирования БД начинается с установления границ будущей разрабатываемой системы и определения функционала, который необходимо реализовать в приложении. Концептуальное проектирование БД 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.

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 [1].

---

[1] Лауреат номинации "**Лучший доклад по UML-моделированию**". Автор доклада награждается книгой Иванова Д. Ю. и Новикова Ф.А. "**Моделирование на UML. Теория, практика, видеокурс**" (www.umlmanual.ru) с автографами авторов

В общем случае логическое проектирование БД заключается в преобразовании концептуальной модели данных в логическую модель данных предприятия с учетом выбранного типа 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.

Последним этапом является физическое проектирование БД, под которым понимается описание конкретной реализации БД, 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 [1].

Физическое проектирование БД предусматривает принятие разработчиком окончательного решения о способах реализации создаваемой базы. Поэтому физическое проектирование обязательно производится с учетом всех особенностей используемой СУБД. Т.к. объектно-ориентированный подход является одним из часто используемых, применяемых при разработке программного обеспечения, было принято решение использовать именно объектно-ориентированную систему управления базами, допускающую создание новых типов данных на основе существующих, поддерживающую продолжительные транзакции и наследование, что способствует ускорению разработки и упрощению сопровождения БД. В нашем случае использована ООСУБД db4objects версии 7.2. 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.

1. Концептуальное проектирование структуры базы данных

В ходе анализа предметной области было выявлено, что необходимо разработать структуру БД, позволяющую сохранять следующее:

- информацию о сотрудниках предприятия;
- информацию об отделах, в которых работают сотрудники;
- информацию о должностях, занимаемых сотрудниками;
- информацию о клиентах, с которыми работает организация;
- информацию об организациях, в которых могут работать клиенты;
- информацию о заявках на оказание услуг, подаваемых клиентами;
- информацию о статусе обработки заявки;
- информацию об оказываемой услуге;
- информацию о типе оказываемой услуги.

На этапе концептуального проектирования информационной модели предприятия, независящей от каких-либо физических условий реализации, необходимо определить типы сущностей, т.е. объекты, информация о которых будет сохраняться в БД.

Список типов сущностей предметной области:

- Service – услуга, оказываемая клиентам;
- ServiceType – тип услуги, оказываемой клиентам;
- Status – статус обработки заявки;
- Client – лицо, подающее заявку на оказание услуг;
- Organization – организация, в которой работает клиент;
- Employee – сотрудник предприятия, оказывающий услуги;
- Position – должность, занимаемая сотрудником на рабочем месте;
- Department – отдел, в котором работает сотрудник;
- Contract – договор об оказании услуг клиенту.

Установим все существующие между выделенными сущностями связи и определим их кратность:

- Связь WorksIn является бинарной и соединяет сущности Organization (0..*) и Client (0..*).
- Связь Takes является бинарной и соединяет сущности Employee (0..*) и Position (1..1).
- Связь WorksAt также является бинарной и соединяет сущности Employee (0..*) и Department (1..1).
- Связь HasType является бинарной и соединяет сущности Service (0..*) и ServiceType (1..1).
- Связь Order является n-арной и соединяет сущности Client (0..*), Service (0..*), Status (0..*), Contract (0..*) и Employee (0..*).

На следующем шаге выполняемого этапа необходимо выявить все данные (атрибуты), описывающие сущности и связи, выделенные в создаваемой модели БД.

Для сущности Client необходимо сохранять атрибуты Name, Address и Phone, включающие соответственно Ф.И.О., адрес и контактный телефон клиента.

В сущности Organization предусмотрено сохранение атрибутов Name, Address и Phone, содержащих информацию соответственно о наименовании, адресе и контактном телефоне организации клиента.

Для сущности Employee необходимо сохранять атрибуты Name, Address и Phone, включающие соответственно Ф.И.О., адрес и контактный телефон сотрудника, принимающего заявку на оказание услуги.

В сущности Position сохраняется атрибут Name, содержащий название должности сотрудника.

Сущность Department включает атрибут Name, содержащий название отдела, в котором работает сотрудник.

В сущности Service необходимо сохранять атрибуты Name, Price и NDS, включающие информацию соответственно о наименовании, цене и сумме НДС с оказываемой услуги.

Сущность Status содержит атрибут Name, описывающий текущую стадию обработки заявки.

В сущности ServiceType сохраняется атрибут Name, хранящий наименование типа оказываемой услуги.

Для сущности Contract необходимо предусмотреть сохранение атрибутов Name и Dates, содержащих информацию о номере договора и дате оказания услуги соответственно.

Для графического представления сущностей и связей создаваемой БД (см. рис.1) разработана ER-диаграмма предметной области, для отображения которой использована нотация унифицированного языка моделирования UML [3]. После того, как построена концептуальная модель БД, необходимо проверить ее на достоверность с помощью определения соответствия транзакциям пользователя. Проверим возможность выполнения следующих основных транзакций:

T1. Добавление нового клиента
T2. Добавление статуса услуги
T3. Добавление новой услуги
T4. Добавление нового типа услуги
T5. Добавление нового сотрудника
T6. Добавление новой должности
T7. Добавление информации о совершенной услуге
T8. Добавление новой организации
T9. Добавление нового договора об оказании услуги
T10. Добавление нового отдела предприятия

Проверим выполнимость транзакций при помощи графического представления путей выполнения транзакции (карты транзакций), изображённой на рис.1.

Для каждой транзакции в круглых скобках указаны операции, выполняемые транзакцией над данной сущностью: I (Insert) – вставка нового экземпляра сущности, U (Update) – обновление значений атрибутов, R (Read) – чтение значений атрибутов, D (Delete) – удаление сущности из БД.

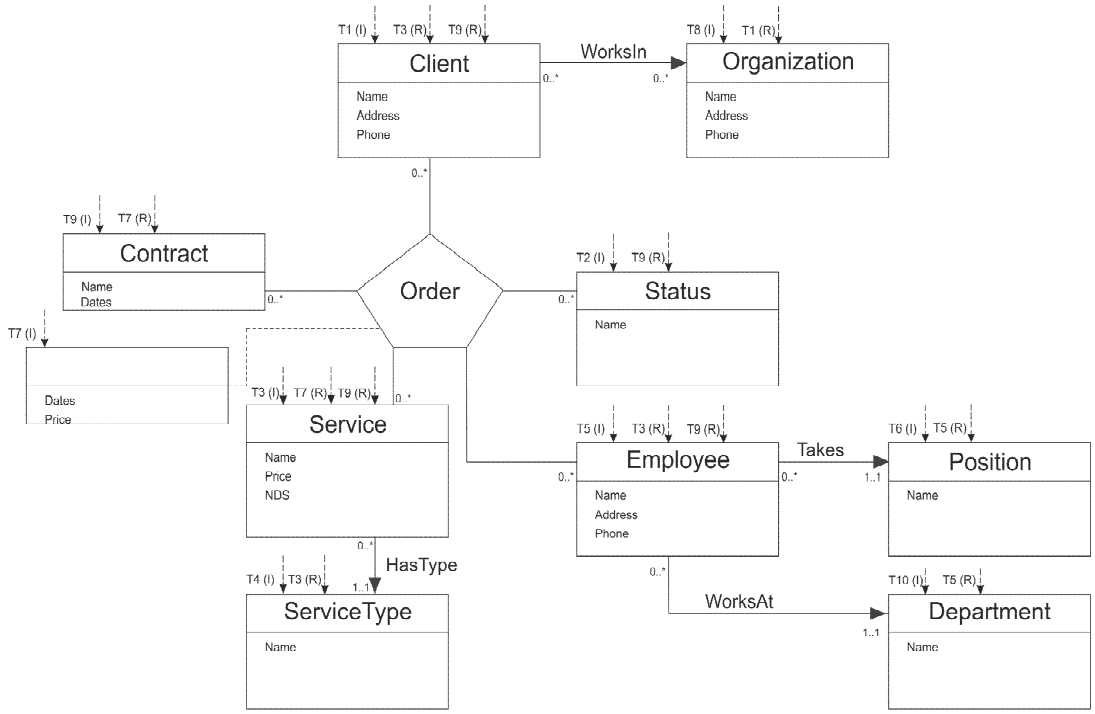


Рис. 1 – Карта выполнения транзакций

Из рис.1 следует, что все описанные ранее транзакции выполнимы, следовательно, концептуальное проектирование структуры БД выполнено верно и можно приступать к логическому проектированию.

2. Логическое проектирование структуры базы данных

Следующим этапом является логическое проектирование БД, которое в общем случае заключается в преобразовании концептуальной модели данных в логическую модель данных предприятия с учетом выбранного типа СУБД (в нашем случае, в понятия объектной СУБД, т.е. в логическую объектно-ориентированную). При этом необходимо выполнить следующие операции:

1. Выполнить обобщение/специализацию;
2. Преобразовать связи с атрибутами в отдельные классы;
3. Преобразовать сложные связи в отдельные классы.

Проанализировав рис. 1, можно утверждать, что все выделенные классы (Client, Organization, Status, Employee, Position, Department, Service, ServiceType и Contract) сходны, так как характеризуются общими атрибутами (в нашем случае названием, атрибутом Name). Поэтому имеет смысл выделить общий базовый абстрактный класс (выполнить процедуру обобщения), содержащий название (например, класс NamedObject), а классы Client, Organization, Status, Employee, Position, Department, Service, ServiceType и Contract сделать производными от NamedObject.

Классы Client и Employee также имеют одинаковые атрибуты (Address и Phone), следовательно, выделим для них базовый класс OrderedContragent.

Выделим также базовый класс OrderedObject для объектов, содержащих заказы, а производными от него станут классы OrderedContragent, Service, Status и Contract.

Также объявим абстрактный базовый класс, содержащий списки работников EmploeesObject, а производными от него будут являться классы Department и Position.

Необходимо ввести промежуточный класс для n-арной связи Order, так как она связывает между собой пять сущностей – клиента, подающего заявку на оказание услуги

(Client), статус обработки заявки клиента (Status), оказываемую клиенту услугу (Service), итоговый контракт с клиентом, свидетельствующий об исполнении услуги (Contract), а также сотрудника, оказывающего услугу (Employee). Результат проделанных операций представлен на рис. 2.

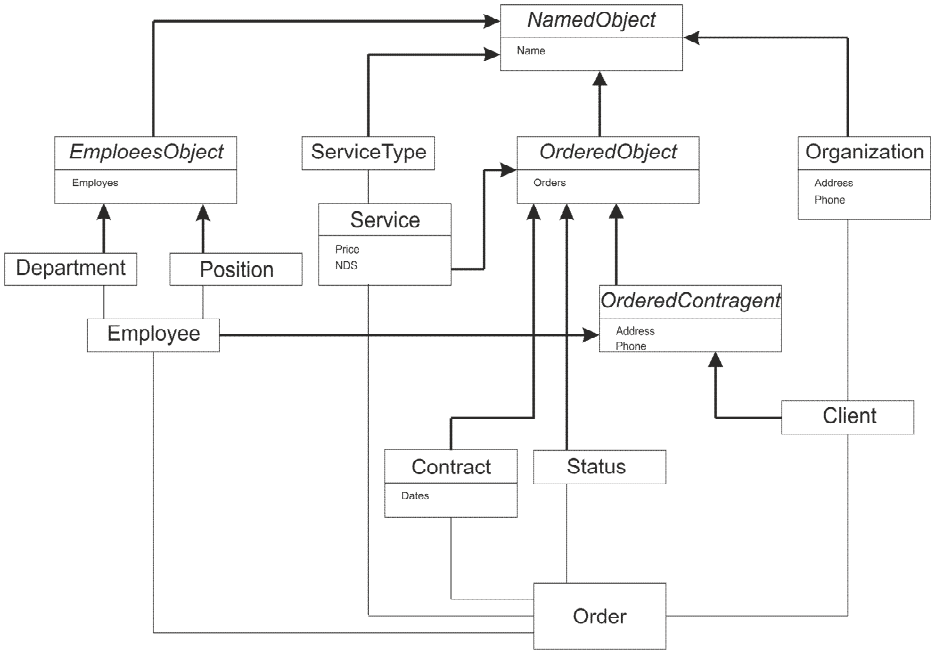


Рис. 2 – Логическая модель

3. Физическое проектирование структуры базы данных

Для разработки структуры БД использована бесплатная ООСУБД db4objects версии 7.2, а сама реализация представляет собой процесс написания классов и организации ассоциаций. Для реализации структуры спроектированной БД в среде целевой СУБД используем интегрированную среду разработки Microsoft Visual Studio 2010. Реализуем БД при помощи языка программирования C#, используя его функциональные особенности (рис.3).

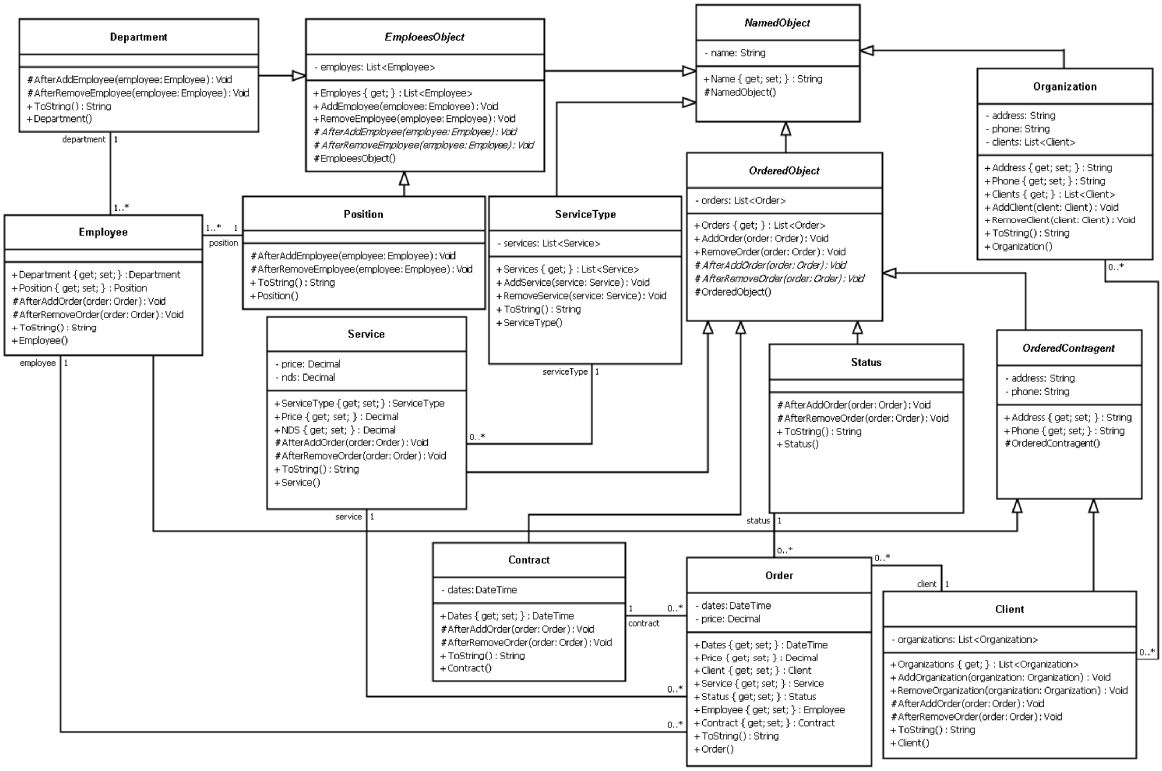


Рис. 3 – Физическая модель структуры базы данных

Для тестирования корректности спроектированной БД реализуем консольное приложение (рис. 4), заполним созданную БД тестовой информацией и проверим разработанное приложение на выполнение. После запуска выполняемого файла пользователю предоставляется диалоговое окно, в котором, выбрав соответствующий пункт меню, можно выполнить все необходимые действия. После выбора одного из пунктов ниже отображается связанная информация. На рис. 4 приведена экранная форма, иллюстрирующая работу созданного приложения.

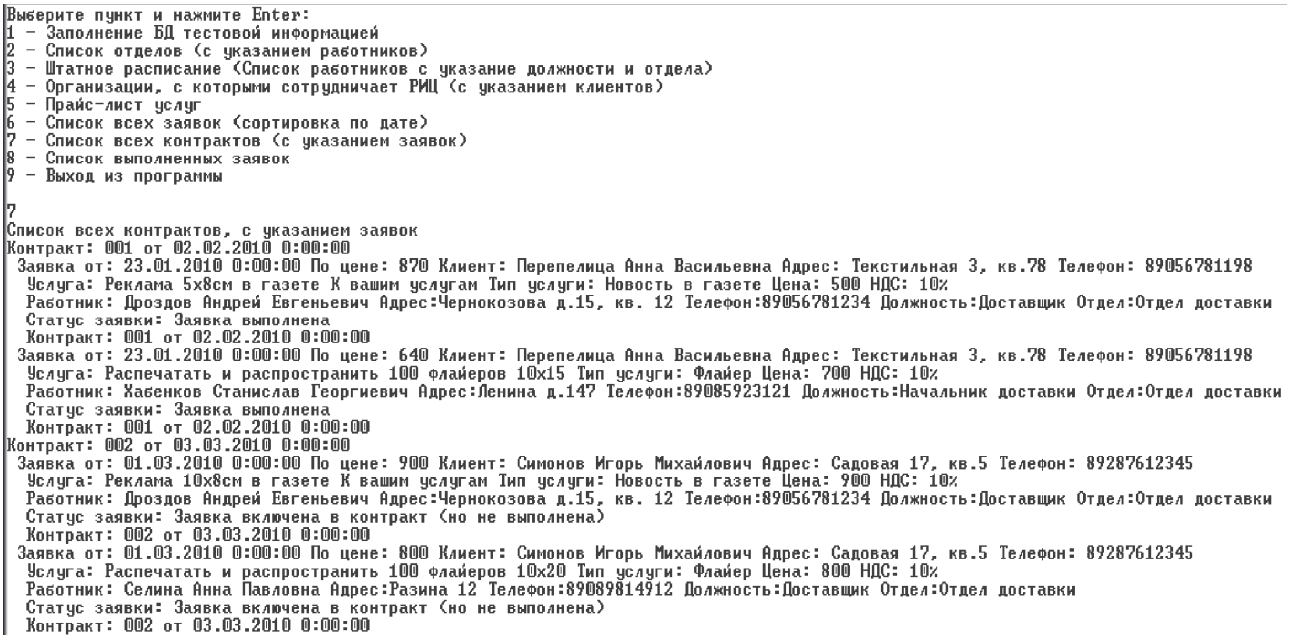


Рис. 4 – Диалоговое окно тестового приложения

Резюмируя вышесказанное, можно сделать вывод, что разработанное приложение полностью удовлетворяет описанным требованиям и позволяет сохранять всю необходимую информацию.

Выводы

На некоторых предприятиях учет оказанных услуг ведется вручную, тогда как внедрение созданной ИС позволит усовершенствовать этот процесс и сделать его намного более быстрым и удобным. Также внедрение поспособствует повышению конкурентоспособности предприятия на рынке, так как в настоящее время множество предприятий уже перешло от ведения учета вручную к использованию ИС, и чтобы не потерять свое место на рынке, необходимо следить за развитием информационных технологий. В статье представлено практическое применение методологии «сущность-связь», что продемонстрировано созданием концептуальной, логической и физической моделей БД.

Литература

1. 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.
2. Дейт К. Дж., Введение в системы баз данных, 7-е издание. : Пер. с англ. – М. : Издательский дом «Вильямс», 2001. – 1072 с. : ил. – Парал. тит. англ.
3. Мюллер Р. Дж., Базы данных и UML проектирование: Пер. с англ. – М. : Лори, 2002. – 420с.: ил. – Парал. тит. англ.

УДК 004.43:378.09

ПРОБЛЕМЫ ИЗУЧЕНИЯ ОБЪЕКТНЫХ ТЕХНОЛОГИЙ В ВУЗЕ[1]

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

Концепция объектно-ориентированного программирования (ООП) подразумевает, что основой управления процессом реализации программы является передача сообщений объектам. Поэтому объекты должны определяться совместно с сообщениями, на которые они должны реагировать при выполнении программы. В этом состоит главное отличие ООП от процедурного программирования, где отдельно определённые структуры данных передаются в процедуры (функции) в качестве параметров. Таким образом, объектно-ориентированная программа состоит из объектов, которые взаимодействуют друг с другом через определённые интерфейсы [4]. Объектно-ориентированный язык программирования должен обладать следующими свойствами:

1. абстракции – формальное представление о качествах или свойствах предмета путем мысленного удаления некоторых частностей или материальных объектов;
2. инкапсуляции – механизма, связывающего вмести код и данные, которыми он манипулирует, и защищающего их от внешних помех и некорректного использования;
3. наследования – процесса, с помощью которого один объект приобретает свойства другого, т.е. поддерживается иерархическая классификация;
4. полиморфизма – свойства, позволяющего использовать один и тот же интерфейс для общего класса действий.

Разработка объектно-ориентированных программ состоит из следующих последовательных работ:

- определение основных объектов, необходимых для решения данной задачи;
- определение закрытых данных (данных состояния) для выбранных объектов;
- определение второстепенных объектов и их закрытых данных;
- определение иерархической системы классов, представляющих выбранные объекты;
- определение ключевых сообщений, которые должны обрабатывать объекты каждого класса;
- разработка последовательности действий, которые позволяют решить поставленную задачу;
- разработка методов, обрабатывающих каждое сообщение;
- очистка проекта, то есть устранение всех вспомогательных промежуточных материалов, использовавшихся при проектировании;
- кодирование, отладка, компоновка и тестирование.

Объектно-ориентированное программирование позволяет программисту моделировать объекты определённой предметной области путем программирования их содержания и поведения в рамках конструкции класса. Конструкция «класс» обеспечивает механизм инкапсуляции для реализации абстрактных типов данных.

Элементы объектно-ориентированного программирования (ООП) появились в конце 60-х годов в языке моделирования Симула, затем получили свое развитие в нескольких других языках программирования. В настоящее время ООП принадлежит к числу ведущих технологий программирования. Основная цель ООП, как и большинства других подходов к программированию, – повышение эффективности разработки программ. В традиционных методах программирования изменение данных или правил и методов обработки часто приводило к необходимости значительного изменения программы. Всякое существенное изменение программы – это большая неприятность для программиста, так как при этом

[1] Лауреат номинации "**Лучший доклад о методах преподавания объектных технологий в ВУЗе**". Автор доклада награждается правом бесплатной публикации одного доклада по данной тематике на следующей конференции

увеличивается вероятность ошибок, вследствие чего возрастает время, необходимое для «доводки» программы. Использование ООП позволяет выйти из такой ситуации с минимальными потерями, сводя необходимую модификацию программы к её расширению и дополнению. Необходимо заметить, что ООП не является панацеей от всех программистских бед, но его ценность как передовой технологии программирования несомненна. Изучение идей и методов ООП может существенно упростить разработку и отладку сложных программ.

В период быстрорастущего количества объектных языков и развития объектно-ориентированных технологий остро встаёт проблема выбора языка для знакомства с основами объектно-ориентированного программирования. К сожалению, в настоящее время для знакомства с азами объектно-ориентированного программирования отсутствует общепризнанный язык, подобный Паскалю, повсеместно используемому для обучения основам программирования. Одним из вариантов может быть язык Java [3], созданный в 1995 году компанией Sun Microsystems, который распространён более других языков, и имеет много преимуществ. Одним из существенных его достоинств является кроссплатформенность, так как исходный код программы компилируется в специальный байт-код, поэтому может быть запущен на любой виртуальной машине Java, независимо от архитектуры. Но в тоже время существенный недостаток данного языка – количество потребляемых ресурсов (памяти, времени процессора и т. д.). Для преподавания в ВУЗе Java подходит, потому что синтаксис данного языка был частично заимствован из языка C++, что удобно для тех студентов, которые для знакомства с программированием изучали языки C/C++. В то же время это язык с нагроможденными синтаксическими конструкциями, который очень сложен в понимании.

Другим вариантом для знакомства с ООП может служить язык Python [5], так как этот язык имеет минималистический синтаксис, частично заимствованный из языка C, поддерживает структурное, объектно-ориентированное, функциональное и аспектно-ориентированное программирование. Для него характерны динамическая типизация, автоматическое управление памятью, полная интроспекция, поддержка многопоточных вычислений и т. д. И Java, и Python имеют сходные характеристики, однако высокоуровневые конструкции в языке Python более удобны для обучения основам ООП. Также существует реализация для JVM (Java Virtual Machine), при которой Python-код может быть запущен на виртуальной машине Java, что удобно, если совместно изучать языки Java и Python.

Python-программы выполняются медленнее, чем программы Java (обычно в 3-5 раз). Однако программы, написанные на Python, также требуют намного меньше времени для разработки. Это можно объяснить тем, что в Python встроены высокоуровневые типы данных, а также Python обладает динамической типизацией. Мощные типы полиморфных списков и словарей Python, для которых богатая синтаксическая поддержка встроена прямо в сам язык, находят применение почти в каждой Python-программе.

Динамическая типизация требует больше обработки во время выполнения, чем статическая, принятая в Java. Рассмотрим выражение (a + b). Python должен сначала исследовать объекты a и b, для того, чтобы выяснить их типы, которые неизвестны во время компиляции. После этого вызывается соответствующая операция сложения, которая может оказаться перегруженным пользователем методом. Язык Java, с другой стороны, может выполнять более эффективное сложение целых, а также чисел с плавающей точкой, но при этом требуются описания переменных a и b, и Java не позволяет перегружать оператор «+» для экземпляров классов, определенных пользователем.

По вышеописанным причинам язык Python лучше подходит как «склеивающий» язык, в то же время Java лучше характеризуется как низкоуровневый язык для реализации. Фактически, если использовать их совместно для разработки одного приложения, то можно получить довольно неплохой результат. Например: компоненты можно реализовывать на Java, а затем использовать в приложениях на Python; Python также полезно использовать для

прототипов компонентов, пока их разработка не «затвердеет» в Java-реализации [2]. Для поддержки такого типа разработки создается реализация Python, написанная на Java. Она позволяет вызывать Python код из Java и наоборот. В этой реализации исходный код Python транслируется в байт-код Java с помощью библиотеки времени выполнения для поддержки динамической семантики Python.

С целью определения языков, наиболее подходящих для обучения основам ООП, был проведен эксперимент: на разных языках ООП был запрограммирован один и тот же алгоритм (сортировка методом пузырька). Сравнение проводилось на основе времени, которое требуется для сортировки 10000 случайных элементов.

C/C++ – 0,330 сек.
Python – 42 сек.
Java – 0,515 сек.
C# – .NET 3.5 – 0,350 сек.

В то же время, если увеличить количество до 100000 элементов, то для Python это будет довольно объемная программа, и нет смысла включать ее в тест.

C/C++ – 27,6 сек.
Java – 37,7 сек.
C# – 32, 5 сек.

Это объясняется тем, что у Python'а динамическая типизация, т.е. у него совсем другие задачи, чем у трех других исследуемых языков. Динами́ческая типиза́ция – приём, широко используемый в языках программирования и языках спецификации, при котором переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной.

Сравнительная характеристика языков ООП по времени исполнения программ на Pentium 4 (1.8Ггц) и производительности (относительно производительности базового кода на C++) показано на рис. 1.

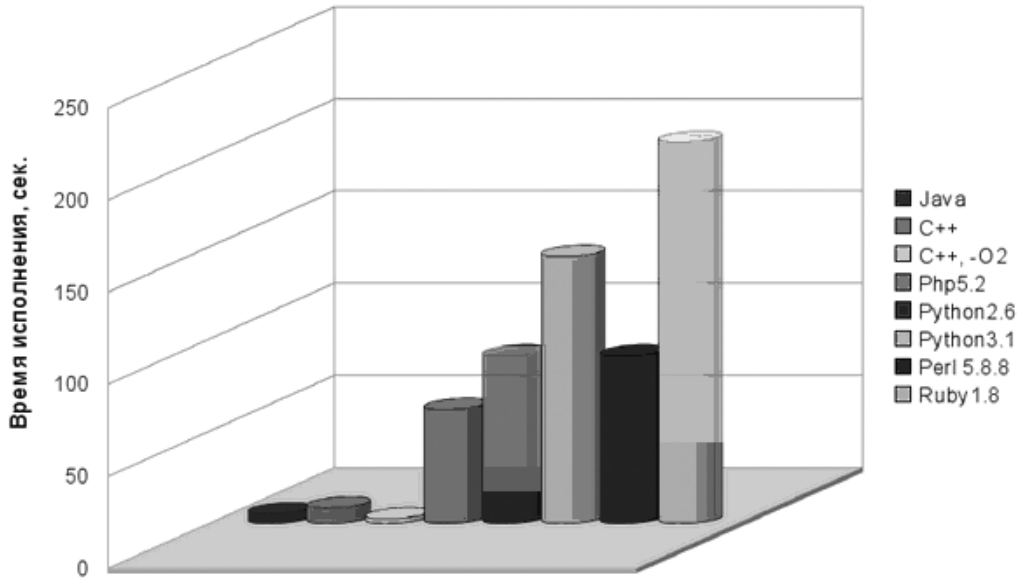


Рис. 1 – Диаграмма производительности языков ООП

Та же диаграмма, но в виде таблицы:

Таблица 1. Сравнительная характеристика языков ООП

Язык программирования	Java	Java - server	C++	C++, -O2	PHP	Python 2.6	Python 3.1	Perl 5.8	Ruby 1.8	Ruby 1.9
Время исполнения, сек	5,3	2,8	8,5	2,6	62	91	145	91	207	30
Производительность, %	160	303	100	327	14	9	6	9	4.11	28

Тестовый программный код

Java, Test01.java

```
public class Test01 {
    public static void main(String[] args) {
        long start = System.currentTimeMillis();
        long r = 0;
        for (int i = 0; i < 10000; i++) {
            for (int j = 0; j < 10000; j++) {
                r = (r + (i * j) % 100) % 47;
            }
        }
        System.out.println("answer: " + r);
        System.out.println("run time (millis): " + (System.currentTimeMillis() -
start));
    }
```

C++, Test01.cpp

```
#include <iostream>
using namespace std;

int main(void) {
    long r = 0;
    for (int i = 0; i < 10000; i++) {
        for (int j = 0; j < 10000; j++) {
            r = (r + (i * j) % 100) % 47;
        }
    }
    cout << "answer: " << r << endl;
}
```

PHP, Test01.php:

```
<?php
$r = 0;
for ($i = 0; $i < 10000; $i++) {
    for ($j = 0; $j < 10000; $j++) {
        $r = ($r + ($i * $j) % 100) % 47;
    }
}
echo 'answer: ' . $r . "\n";
?>
```

Python, Test01.py:

```
r = 0

for i in range(0, 10000):
    for j in range(0, 10000):
        r = (r + (i * j) % 100) % 47

print("answer: ", r)
```

Среда тестирования:

$ cat /proc/cpuinfo |fgrep 'model name'
model name : Intel(R) Pentium(R) Dual CPU T3400 @ 2.16GHz

model name : Intel(R) Pentium(R) Dual CPU T3400 @ 2.16GHz

$ java -version
java version "1.6.0_20"

OpenJDK Runtime Environment (IcedTea6 1.9.7) (6b20-1.9.7-0ubuntu1)
OpenJDK 64-Bit Server VM (build 19.0-b09, mixed mode)

$ g++ -v
gcc version 4.4.5 (Ubuntu/Linaro 4.4.4-14ubuntu5)

$ php -v
PHP 5.3.3-1ubuntu9.3 with Suhosin-Patch (cli) (built: Jan 12 2011 16:07:38)

Copyright (c) 1997-2009 The PHP Group
Zend Engine v2.3.0, Copyright (c) 1998-2010 Zend Technologies

$ python2.6
Python 2.6.6 (r266:84292, Sep 15 2010, 16:22:56)

$ python3.1
Python 3.1.2 (release31-maint, Sep 17 2010, 20:27:33)

На данный момент языков для ООП множество, и у каждого есть свои преимущества и свои недостатки. Для изучения в ВУЗе наиболее всего подходит язык Java параллельно с Python, именно поэтому акцент был сделан на этих двух языках.

Литература

1. Бертран Мейер. Объектно-ориентированное конструирование программных систем + CD. Интернет-университет информационных технологий – ИНТУИТ.ру, Русская Редакция, 2005
2. http://habrahabr.ru
3. http://ru.wikipedia.org/wiki/Java
4. http://refu.ru/refs/67/15499/1.html
5. http://4programmers.org.ua.

УДК 004.4

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ МОДЕЛИРОВАНИЕ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ ИНТЕГРАЦИИ ОТКРЫТЫХ ВИРТУАЛЬНЫХ ЛАБОРАТОРНЫХ КОМПЛЕКСОВ[1]

Жукова Светлана Александровна, к.т.н., докторант, Чайковский технологический институт (филиал) Ижевского государственного технического университета, Россия, Чайковский, otdel_it@chti.ru
Германюк Денис Евгеньевич, аспирант, Чайковский технологический институт (филиал) Ижевского государственного технического университета, Россия, Чайковский

В настоящее время в России активно ведутся работы по созданию и развитию научно-образовательной среды с использованием Интернет–технологий для поддержки процессов проведения научных исследований и образования [1]. Формирование научно-образовательной среды предполагает разработку и внедрение информационных систем, обеспечивающих тесное взаимодействие научных и образовательных ресурсов посредством глобальных вычислительных сетей, и использует методы и технологии построения распределенных открытых систем [2]. Для данных систем характерно:

- территориальная удаленность пользователей;
- использование неоднородного программного и аппаратного обеспечения;
- динамическое наращивание функциональных сервисов приложений;
- множество объектов и их распределенность;
- неоднородность ресурсов,

что и характеризует их как сложные информационные системы. В этом случае возрастает роль моделирования и исследования структуры системы.

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

Рассмотрим моделирование распределенной открытой системы на примере автоматизированной системы интеграции открытых виртуальных лабораторных комплексов (АС ОВЛК). Работа ведется Ижевским государственным техническим университетом в рамках ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009-2013 годы работы по теме «Разработка модели автоматизированной системы интеграции открытых виртуальных лабораторных комплексов (АС ОВЛК)». Открытый виртуальный лабораторный комплекс (ОВЛК) представляет собой систему интеллектуальных, организационных и вычислительных ресурсов, построенную в соответствии с принципами технологии открытых систем и предназначенную для исследования в области образования, науки и бизнеса [2].

Таблица 1. Основные требования АС ОВЛК и тактики их достижения

Наименование требования	Реализация	Тактика
Мобильность, т.е. система должна обеспечивать возможность взаимодействия серверных компонентов, независимо от программно-аппаратной платформы.	Создание механизмов для возможности использования ВЛК разными категориями пользователей на базе различной аппаратуры.	Создание компонентной архитектуры в соответствии со спецификацией J2EE. Построение на основе открытых стандартов (J2EE, XML и т.д.), с возможностью работы с различными Web-серверами.
Модифицируемость, т.е. система должна предусматривать добавление новых компонентов по мере их разработки, с минимальными усилиями.	Использование готовых шаблонов, создание библиотеки шаблонов для исследовательских задач.	Уменьшение связности и увеличения сцепления; повторность использования; сокращение числа модулей; обобщение модуля; разработка интерфейсов для подключения ОВЛК; введение интерфейсов для подключения новых компонентов.
Интероперабельность, т.е. система должна обеспечивать информационную и программную совместимость с внешними системами; возможность подключения виртуальных лабораторий от разных производителей.	Разработка механизмов обмена научной и образовательной информацией: способов и методов обмена. Разработка механизмов настройки интерфейса с внешними системами.	Информационная закрытость: разделение информации на приватную и публичную; разделение методов. Введение нового класса – адаптера. Введение интерфейсов для подключения новых компонентов. Построение компонентов на основе открытых стандартов.
Производительность, т.е. система должна корректно обрабатывать запросы пользователей численностью до нескольких сотен без потери производительности.	Создание механизмов повышения производительности предоставления услуг за счет планирования ресурсов для выполнения заявок	Управление вычислительными ресурсами путем ввода планирования использования, увеличение ресурсов путем горизонтального масштабирования.

Интеграция виртуальных лабораторных комплексов для проведения дистанционных экспериментов с компьютерными моделями и управление ими осуществляется с помощью автоматизированной системы. АС ОВЛК имеет архитектуру распределенных объектов и предназначена для настройки виртуальных лабораторных комплексов с целью их включения в информационное пространство. Основные задачи АС ОВЛК: накопление, хранение, обработка и защита сведений об экспериментах и ВЛК, тестирование ВЛК на соответствие техническим требованиям, настройка интерфейса взаимодействия ВЛК с пользователем, обеспечение удаленного доступа пользователям к сервисам ВЛК, распределение вычислительной нагрузки между ВЛК.

Приведенные в работе [2] концептуальные особенности формирования ВЛК для исследовательских задач позволили сформулировать основные требования АС ОВЛК и тактики их достижения (Таблица 1). Тактика – это проектное решение, которое влияет на достижение определенного (или нескольких показателей) требования. Совокупность принятых тактик позволяет выработать архитектурную стратегию системы.

Проектирование АС ОВЛК как сложной информационной системы выполняется построением ряда моделей, которые описывают структуру системы с разных точек зрения и позволяют определить основные проектные решения, удовлетворяющие ключевым требованиям АС ОВЛК.

Определим задачу моделирования АС ОВЛК как задачу синтеза оптимальной структуры, максимально удовлетворяющей ключевым требованиям. Проблема синтеза структуры АС ОВЛК включает: декомпозиция системы на уровни и подсистемы, определение состава подсистем, распределение обязанностей между программными компонентами, выбор варианта размещения подсистем на физических узлах. Таким образом, задачу синтеза структуры можно описать, как поиск оптимального отображения функционала системы на элементы: программные компоненты и их размещение на распределенных вычислительных узлах.

Формально задачу синтеза структуры системы можно описать как задачу формирования графа *G(E,V),* удовлетворяющего функции *F*

$F(\alpha_i \cdot R_i) \to \max$,

где *F*- интегральный показатель достижения требований $R_{i,}$,

α_i – весовой коэффициент, учитывает приоритет требования,

V- элементы структуры (программные компоненты),

E- связи между элементами структуры.

Дуги графа *G(E,V)* отражают зависимости или отношения между классами и ориентированы в соответствии с сообщениями и информационными потоками, передаваемые от класса к классу.

Основным способом моделирования системы выбран объектно-ориентированный метод (ООМ). Объектно-ориентированное моделирование и проектирование – это подход к решению задач с использованием моделей, основанных на понятиях реального мира. Фундаментальным элементом является объект, объединяющий структуру данных с поведением. В настоящее время посвящено множество работ, описывающих ООМ и соответствующие языки описания моделей (UML), а также примеры использования [3,4].

Задачи моделирования и анализ инструментов ООМ [1] позволили сформулировать основные этапы синтеза структуры АС ОВЛК, определить перечень моделей и соответствующих диаграмм, их назначение и цели анализа (таблица 2).

По результатам выполнения этапов синтеза структуры получено описание архитектуры АС ОВЛК в следующих представлениях: модель декомпозиции, модель размещения.

В качестве архитектурного шаблона выбрана модель распределенного приложения Model-View-Controller. Характерной чертой таких приложений является логическое разделение приложения на две и более частей, каждая из которых может выполняться на

отдельном компьютере. Выделенные части приложения взаимодействуют друг с другом, обмениваясь сообщениями в заранее согласованном формате. В этом случае двухзвенная архитектура клиент-сервер становится трехзвенной, а в некоторых случаях, она может включать и больше звеньев так как позволяет распределить компоненты системы между вычислительными узлами и максимально разгрузить клиента. На рисунке 1 представлена клиент-серверная архитектура системы.

Таблица 2. Основные этапы синтеза структуры АС ОВЛК

Наименование этапа	Наименование модели/диаграммы	Цель анализа модели
Постановка задачи	Модель требований Use-case	формирование ключевых функциональных требований АС ОВЛК; список пользователей; нефункциональные требования;
Проектирование структуры	Модель анализа Диаграмма последовательности	анализ групп операций, повторяющихся в нескольких классах
	Диаграмма классов	анализ групп атрибутов, повторяющихся в нескольких классах, анализ классов играющих одинаковую роль, оценка связности между классами на основе: анализ ассоциации, анализ сообщений между классами
	Модель проекта Диаграмма взаимодействия; диаграмма размещения диаграмма классов	анализ зависимости между пакетами, анализ возможности размещения пакетов на распределенных узлах; анализ возможности планирования ресурсов
	Диаграмма последовательности	анализ механизмов реализации защиты, хранения в БД, механизмы взаимодействия в распределенными подсистемами, возможность применения готовых паттернов для реализации механизмов
Исследование структуры	Модель реализации диаграмма размещения	Расчет показателей структуры и их оценка на возможность достижения ключевых требований АС ОВЛК

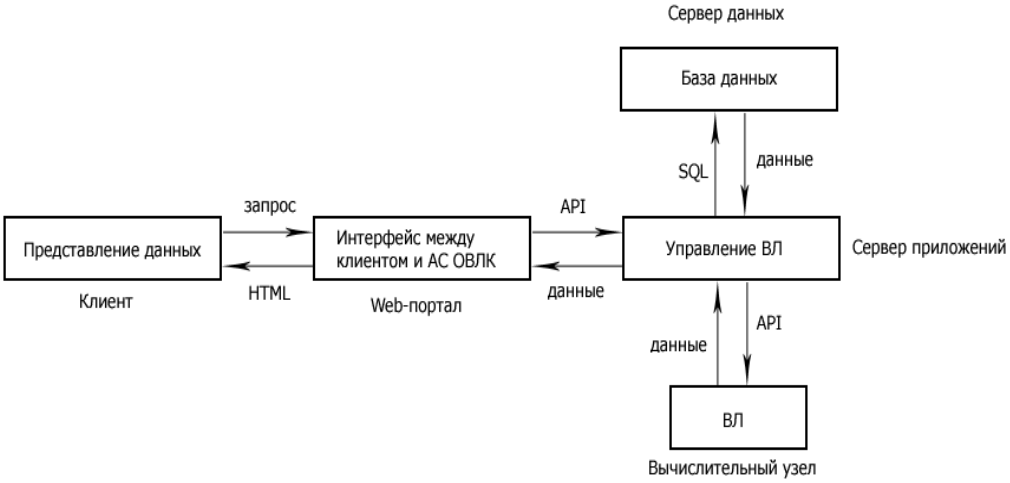


Рис. 1 – Клиент-серверная архитектура АС ОВЛК

Она построена на основе трёхзвенной архитектуры с добавлением звена по управлению вычислительными лабораториями и звена «Вычислительный узел», обеспечивающего проведение эксперимента.

На рисунке 2 представлена модульная структура АС ОВЛК в виде модели декомпозиции, построенная на основе модели требований к системе [2]. АС ОВЛК состоит из подсистем, логически объединенных в пакеты.

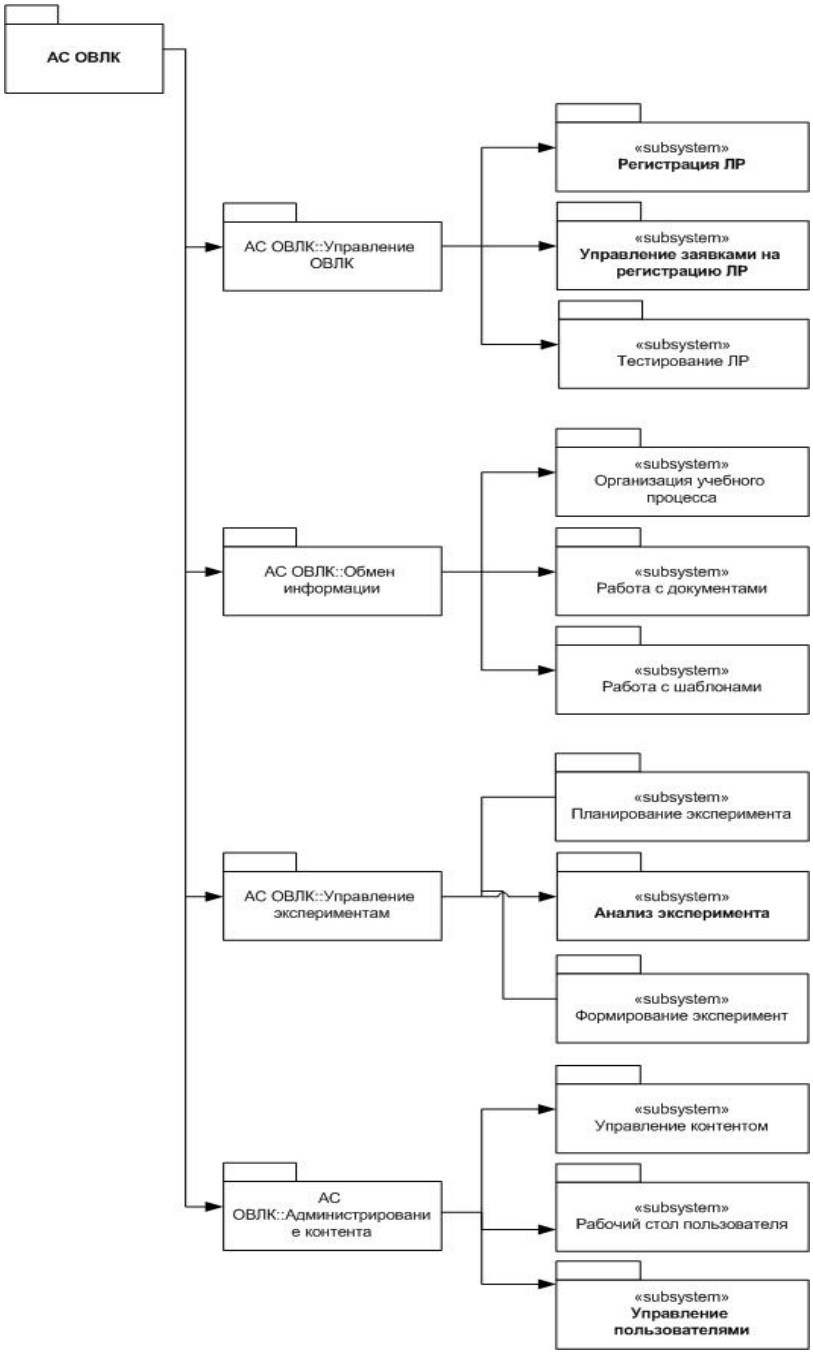


Рис. 2 – Модель декомпозиции АС ОВЛК

Каждый пакет включает подсистемы, которые выполняют строго отведённые им роли:

- пакет управления открытыми виртуальными лабораторными комплексами предназначен для выполнения функций, связанных с управлением открытым виртуальным лабораторным комплексом: регистрация ВЛ и прочих ЛР, поиск, тестирование, подключение к ИП, предоставление доступа к ЛР пользователей;

- пакет обмена информацией предназначен для организации электронного документооборота и обмена исследовательскими данными между внешними системами;
- пакет управления экспериментами предназначен для формирования ВЛК в соответствии с целью исследования по запросу пользователя, формирования данных для экспериментов, планирования (распределения вычислительных ресурсов между ВЛК) и его выполнение, анализ полученных результатов;
- пакет администрирования контента служит для обеспечения доступа к ресурсам ОВЛК, публикации информации о лабораторных ресурсах на сайте. Включает набор функций по наполнению, обслуживанию и публикации информационного портала.

На рисунке 3 показана диаграмма развёртывания, которая представляет собой распределение компонентов системы между вычислительными узлами:

- сервер приложений – ядро АС ОВЛК, программно-аппаратный комплекс, на котором размещены основные подсистемы. Сервер приложений управляет процессами выполнения необходимых расчётов для проведения исследований, вычислительными ресурсами, а также обрабатывает заявки клиента.
- веб-портал – сервер, на котором установлено Web-приложение, обеспечивающее взаимодействие пользователя и АС ОВЛК, принимает запросы пользователя и возвращает результаты обработки запросов, на данном сервере размещена подсистема управления контентом.
- вычислительный ресурс – сервер либо рабочая станция, предназначен для выполнения расчётов, задание на проведение которых выдает сервер приложений.
- информационное хранилище – сервер, предназначенный для хранения ресурсов ОВЛК (информационных, организационных)

клиент – компьютер пользователя, инициирующий запросы на проведение экспериментов и получение результатов эксперимента.

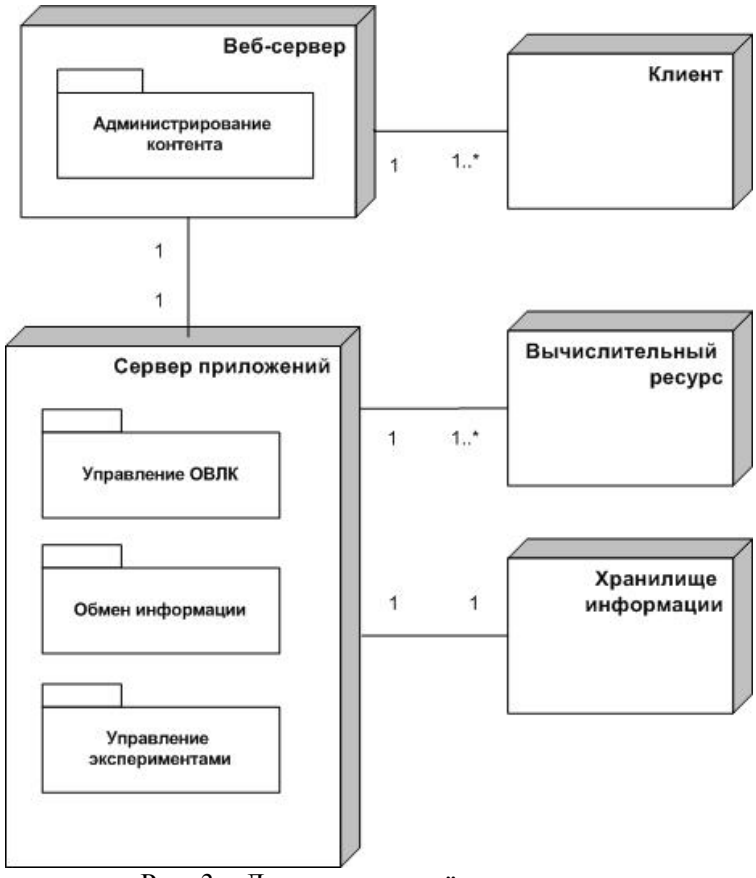


Рис. 3 – Диаграмма развёртывания системы

Интеграция разнородных ресурсов, направленных на решение исследовательских задач в области науки и образования, выдвигает требования к информационным системам: мобильность, интероперабельность, модифицируемость. Такие системы являются достаточно сложными и их проектирование осуществляется путем моделирования и исследования характеристик структуры, что позволяет значительно снизить риски создания АС и максимально удовлетворить ключевые требования.

Преимущество объектно-ориентированного подхода оказывается большим, чем может показаться с первого взгляда. Выявление объектов, изучение его свойств (статические модели) и поведения с другими объектами (динамические модели) позволяет разработчику формировать и оценивать проектные решения по структуре системы на этапе предварительного проектирования. ООМ обладает достаточно мощным инструментом построения распределенных открытых систем и позволяет моделировать системы для исследовательских задач с учетом заявленных требований.

Литературы

1. S.A. Zhukova, S.Zh. Kozlova, I.N. Efimov, E.A. Ermolaeva Technologies of development of the open systems and their application in science and education: the review of papers and development perspective (Технологии построения открытых систем в формировании научно-образовательной среды: обзор работ и перспективы развития) // 12 международная конференция Computer Science and Information Technologies (CSIT'2010) (Информатика и Информационные технологии), Уфимский государственный авиационный технический университет (УГАТУ), С. 208-212.
2. Разработка модели автоматизированной системы интеграции открытых виртуальных лабораторных комплексов. Этап 2: «Построение концепции автоматизированной системы открытых виртуальных лабораторных комплексов» (промежуточный)» / Отчет о научно-исследовательской работе (промежуточный)/ Ижевский государственный технический университет; рук. И.Н.Ефимов. – Ижевск, 2010. – 199 с. – ГР № 02.740.11. 0658
3. Арлоу, Нейштадт. UML 2 и Унифицированный процесс: практический объектно-ориентированный анализ и проектирование. – Символ-Плюс, 2-е изд. – 2007. – с. 624
4. К. Ларман. Применение UML 2.0 и шаблонов проектирования. – Вильямс, 3-е изд. – 2007. – с.730

УДК 004.457

МОДЕЛИРОВАНИЕ ПОВЕДЕНИЯ ОБЪЕКТОВ С ПОМОЩЬЮ КОНЦЕПЦИИ КОНЕЧНЫХ АВТОМАТОВ[1]

Галиаскаров Эдуард Геннадьевич, к.х.н., доцент, доцент, Ивановский государственный химико-технологический университет, Россия, Иваново, galiaskarov@isuct.ru

Особенности дискретно-детерминированного подхода чаще всего рассматриваются на базе теории автоматов. Теория автоматов – раздел теоретической кибернетики, в котором изучаются математические модели – автоматы. Автомат перерабатывает дискретную информацию и меняет свои внутренние состояния в допустимые моменты времени. Если конечное множество объектов можно интерпретировать как алфавит, а сами объекты рассматривать как буквы этого алфавита, набор из этих букв представляет собой упорядоченную совокупность, называемую словом, то автомат представляет собой некий черный ящик, который преобразует буквы входного сигнала в буквы выходного сигнала. Конечным автоматом будем называть такой автомат, у которого множество состояний конечно [1]. Для того чтобы задать конечный автомат (finite automata), необходимо задать одношаговую функцию перехода

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

$$z(t) = \varphi[z(t-1), x(t)]$$

и функцию выхода

$$y(t) = \psi[z(t-1), x(t)].$$

Автомат задается F-схемой вида:

$$F = \langle Z, X, Y, \varphi, \psi, Z_0 \rangle,$$

где Z –конечное множество состояний; X – входной сигнал (алфавит); Y – выходной алфавит; φ, ψ - функции перехода и выхода; Z0 – начальное состояние алфавита

Автомат функционирует в дискретные моменты времени, которые называются тактами.

Любой автомат может быть задан с помощью таблиц переходов и выходов. Автомат также может быть задан с помощью ориентированного графа. В последнем случае это легко сделать на UML.

Рассмотрим довольно простую, но достаточно иллюстративную задачу. Предположим, что необходимо сконструировать автомат для продажи билетов. Требования к системе просты. Этот автомат принимает монеты достоинством 1, 2, 3 и 5 руб. и выдает билеты стоимостью 5 руб.

Сначала опишем данное устройство в каноническом виде. Данное устройство можно представить как конечный асинхронный автомат Мили с множеством состояний Z ={0, 1, 2, 3, 4}, где каждое состояние представляет собой сумму монет в монетоприемнике, причем Z = 0 означает отсутствие монет в монетоприемнике как в режиме ожидания, так и в режиме выдачи билета, когда сумма монет превысит 4, входным алфавитом X = {1, 2, 3, 5} и выходным алфавитом Y = {0, 1}, где 0 соответствует ситуации "билет не выдается", а 1 – ситуации "билет выдается". Функция переходов φ(t) определяется соотношением $z(t) = (z(t-1) + x(t)) \bmod 5$

а функция выходов ψ(t) – соотношением

$$y(t) = \begin{cases} 0, & когда \quad z(t-1) + x(t) \le 4 \\ 1, & если \quad z(t-1) + x(t) > 4 \end{cases}$$

В качестве иллюстрации построим таблицы переходов и выходов рассматриваемого автомата:

Таблица переходов

x	Z				
	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	0
3	3	4	0	0	0
5	0	0	0	0	0

Таблица выходов

x	z				
	0	1	2	3	4
1	0	0	0	0	1
2	0	0	0	1	1
3	0	0	1	1	1
5	1	1	1	1	1

Формализуем описание рассматриваемой системы с помощью UML [2].

Использование автомата для продажи билетов исключительно просто и представляется одним единственным вариантом использования (рис. 1).

Типичный процесс приобретения билета можно отразить на диаграмме деятельности (рис. 2). В соответствии со сценарием использования и условием задачи структурную модель автомата для продажи билетов можно изобразить на диаграмме классов, представленной на рис. 3.

Как следует из математического описания, рассматриваемую систему можно представить в виде асинхронного автомата Мили. Введем некоторые обозначения для упрощения изображения параметров на диаграммах состояния:

X – номинал монеты;

Z – сумма денег в монетоприемнике;

L – размер сдачи;
Y – выдан или нет билет

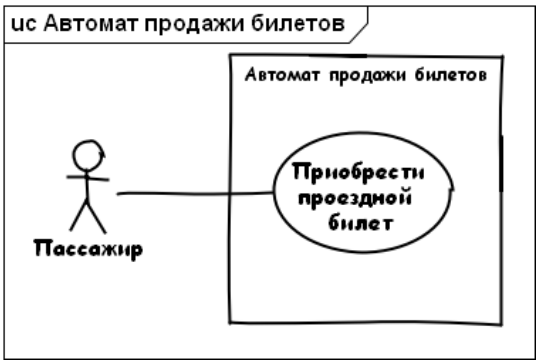


Рис. 1 – Диаграмма использования автомата продажи билетов

Согласно исходной постановке задачи, т.е. автомату Мили, автомат для продажи билетов в процессе деятельности находится в одном из пяти возможных состояниях, представленных на рис. 4.

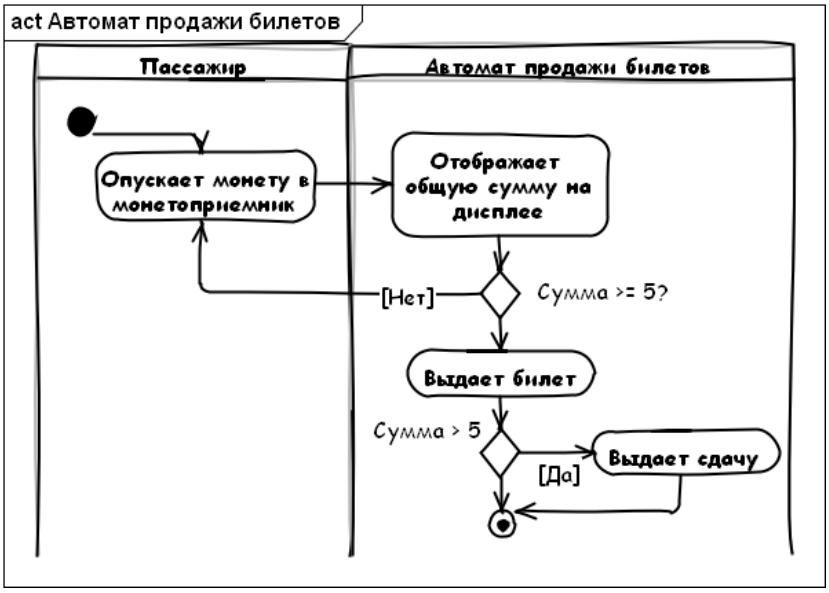


Рис. 2 – Типичный сценарий использования автомата продажи билетов

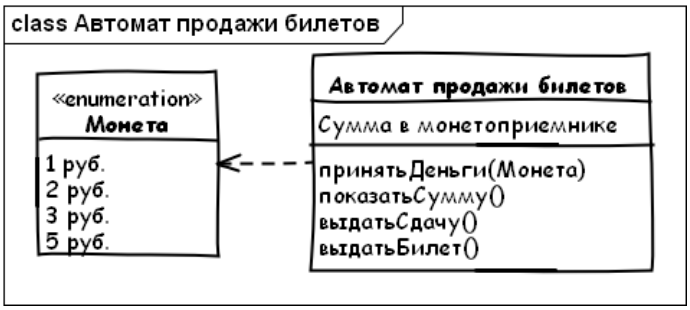


Рис. 3. Диаграмма классов автомата продажи билетов

Диаграмма получилась довольно большой и сложной. А что будет, если к монетам добавится возможность использовать и бумажные банкноты? Очевидно, что диаграмма значительно усложнится. Можно ли найти иной способ отображения поведения автомата? Да, такой способ есть. Опишем автомат через два состояния: состояние S0 – прием монет, и состояние S1 – выдача билета. Таким образом, начальную постановку задачи в виде автомата Мили, мы заменили автоматом Мура, у которого выходная функция зависит только от состояния. Входная переменная X может принимать значение 1, 2, 3 или 5. Выходная переменная Y – 0 или 1. Введем внутреннюю переменную Z – сумма денег в монетоприемнике. Переход из

состояния S0 в состояние S1 происходит по условию (Z >= 5). В начальный момент времени система находится в состоянии S0, а сумма денег Z = 0.

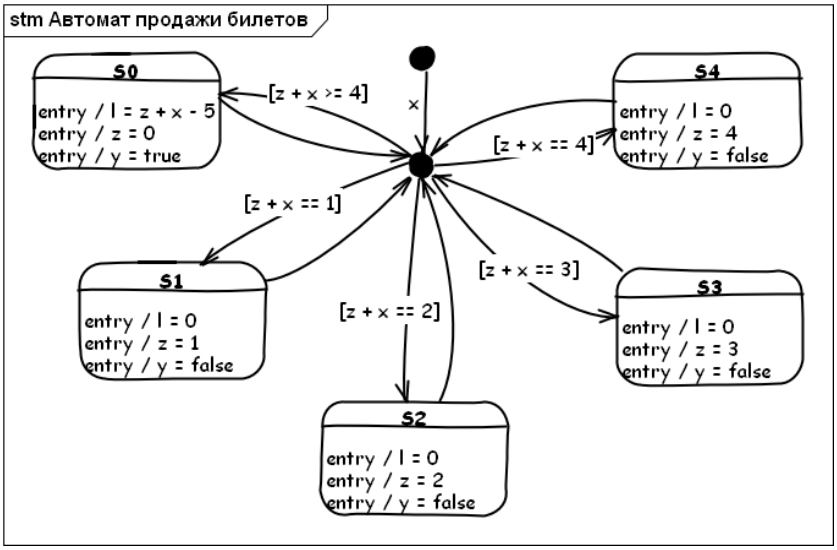


Рис. 4 – Исходный вариант диаграммы состояний автомата продажи билетов

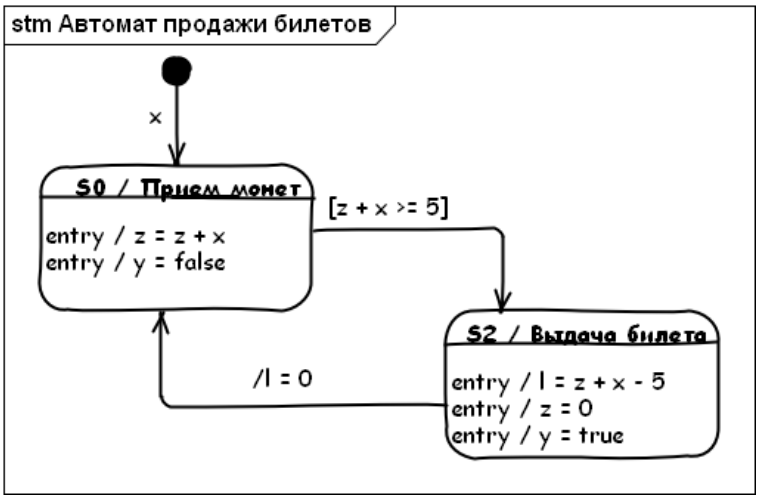


Рис. 5 – Вариант диаграммы состояний автомата продажи билетов как автомата Мура

Для того, чтобы убедится в корректности предложенных моделей, можно реализовать эти модели в замечательном инструменте Matlab и провести имитационное моделированиу. С этой целью необходимо использовать toolbox Simulink, предназначенный для визуального моделирования самых разнообразных систем. В состав Simulink входит другой замечательный инструмент Stateflow, предназначенный для описания и моделирования дискретных систем на базе теории конечных автоматов [3]. При этом используемая в Stateflow графическая система обозначения Харела практически полностью соответствует синтаксису и семантике построения диаграмм состояний UML [2], что позволяет без труда перенести результаты UML моделирования в модель Simulink.

Автор выражает благодарность за полезные дискуссии Денису Юрьевичу Иванову, одному из авторов книги «Моделирование на UML».

Литература

1. Советов Б.Я., Яковлев С.А. Моделирование систем: Учеб. Для вузов – 3-е изд., перераб. и доп. – М.: Высш. шк., 2001. – 343 с.: ил..
2. Новиков Ф.А., Иванов Д.Ю. Моделирование на UML. Теория, практика, видеокурс. – СПб.: Профессиональная литература, Наука и техника, 2010. – 640 с.: ил.
3. Гультяев А.К. MATLAB 5.2. Имитационное моделирование в среде Windows: Практическое пособие. – СПб.: КОРОНА принт, 1999. – 288 с.

УДК 004.415.2(06)

ИСПОЛЬЗОВАНИЕ ПАТТЕРНА ПРОЕКТИРОВАНИЯ MODEL-VIEW-CONTROLLER ПРИ РАЗРАБОТКЕ WEB-ПРИЛОЖЕНИЙ

Беликова Наталья Валентиновна, к.т.н., доцент, Шахтинский институт (филиал) Южно-Российского государственного технического университета (Новочеркасского политехнического института), Россия, Шахты, bnatv@yandex.ru

Галич Марк Геннадьевич, студент, Шахтинский институт (филиал) Южно-Российского государственного технического университета (Новочеркасского политехнического института), Россия, Шахты, mark@galich.me

Паттерны проектирования – это шаблоны, адаптированные под решение наиболее общих задач, основное предназначение которых – описание взаимодействия объектов в приложении. Паттерн не является законченным образцом, и не может быть прямо преобразован в код; это лишь вспомогательный каркас для задачи, который может существенно упростить разработку приложения. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться [1].

Большинство разработчиков в наше время применяют объектно-ориентированный подход в программировании при работе над крупными проектами, или хотя бы знакомы с его основной концепцией. Трудно, а порой даже не возможно, начинать серьезную работу с нуля, не используя наработок в коде или хотя бы в проектировании. В связи с этим, у программистов появляется потребность выделять для себя ключевые моменты из каждого проекта и с каждым разом, абстрагируясь от каких-то конкретных задач, строить каркас для будущих разработок. Такой подход удобен при индивидуальной работе над проектом. Если же разработку ведёт команда, то работу будет задерживать необходимость вникать в выработанный годами подход. Остаётся или тратить огромное количество времени на изобретение очередного велосипеда, или же начать использовать различные паттерны проектирования.

Паттерн проектирования Model-View-Controller (Модель-Представление-Контроллер) лег в основу архитектурного решения первой среды программирования с графическим интерфейсом пользователя – Smalltalk-80. Впервые MVC описал еще в 1978 году норвежец Трюгве Ринскауг, работавший некоторое время в лаборатории Xerox PARC [3]. Реализацию шаблона в Smalltalk-80 описал несколько позже Стив Бурбек [4].

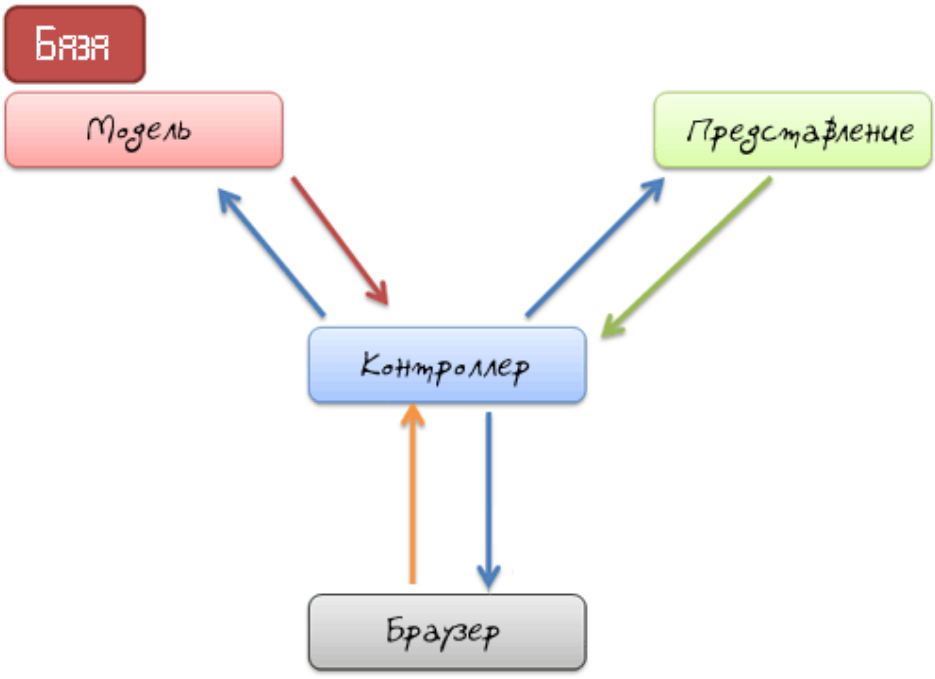


Рис. 1 – Архитектура Model-View-Controller

Архитектура Model-View-Controller позволяет разделить модель данных, пользовательский интерфейс и управляющую логику на три отдельные составляющих (Рис. 1). Причем разделить так, что изменение одного компонента не повлияет на работу остальных. Использовать эту архитектуру особенно удобно при командной работе над большими проектами. Например, при разработке web-приложения, можно отделить разработку части сайта, отвечающей за обработку данных, абсолютно не беспокоясь о том, как эти данные будут представлены, т.к. над интерфейсом приложения могут абсолютно независимо работать верстальщики и дизайнеры. Что же касается программистов, то они также могут разделять свою работу – например, часть разработчиков может сосредоточиться на разработке логики программы, а другая – на проектировании структуры базы данных. Из описанных выше примеров видно, что при использовании MVC становится возможным распределить работу команды, тем самым повысив общую производительность.

MVC состоит из следующих сущностей [2]:

- Модель данных (Model) в целом понимается как компонент администрирования данных. В большинстве случаев это общение с базами данных – выборка определённых данных, добавление новых и изменение существующих. Модель не знает, в каком виде надо представлять запрашиваемые у неё данные пользователю, она лишь совершает операции с «сырыми» данными. Она должна иметь стандартизированный универсальный интерфейс, чтобы одну модель могло использовать любое количество других подсистем любое количество раз.
- Вид (View) описывает формат вывода данных пользователю, то есть визуальное представление данных, взятых из Модели. Он получает на входе набор сырых данных, которые необходимо передать пользователю, форматирует их в соответствии с требованиями пользователя (на примере веб-разработок – одна и та же страница может быть представлена как в виде HTML-документа, так и в XML, JSON и т.д.). Главное удобство системы видов состоит в том, что для каждого конкретного случая представления данных не надо описывать шаблон целиком, можно скомпоновать его из нескольких стандартных блоков и одного, уникального для этого случая. Это избавляет от проблем с внесением изменений в шаблон, когда требуется изменить, допустим, заголовок сайта на всех его страницах (если все страницы сайта используют один Вид заголовка, достаточно изменить лишь его, а не менять вручную шаблон каждой страницы).
- Контроллер (Controller) осуществляет управление всеми процессами приложения. Он обрабатывает внешние запросы и выполняет содержащиеся в этих запросах требования, раздавая соответствующие указания Моделям, и передает полученные от Моделей данные соответствующим ситуации Видам для корректного вывода нужной информации.

В целом, работу MVC можно представить так:

- команда (уведомление о нажатии кнопки, запроса адреса сайта и т. д.) передается Контроллеру;
- Контроллер, исходя из полученных данных, определяет и вызывает Модель;
- Модель, на основе заложенной в нее бизнес-логики, формирует набор данных;
- Контролер выбирает Представление и связывает его с данными (Моделью);
- Представление отображает данные пользователю.

К основным минусам паттерна MVC можно отнести:

- Увеличение объема кода по сравнению с программированием без применения каких-либо паттернов;
- Необходимость соблюдения заранее заданного интерфейса;
- Для поддержки разработки требуются более квалифицированные специалисты.

К главным плюсам отнесём следующее:

- Более гибкий код;
- Возможность повторного использования каждой из трёх составных частей MVC;

- Безболезненная замена и изменение каждого из элементов системы (другие алгоритмы расчета, способы хранения данных, визуальные представления и т.д.);
- Скорость разработки;
- Легкость поддержки кода.

В современном мире нет ничего статичного, и, пожалуй, в наибольшей степени это касается программного обеспечения. Объемы и сложность современных приложений (как среди «настольного» ПО, так и среди серверного и веб-сервисов) растут с пугающей скоростью. Любое приложение профессионального уровня представляет собой сложную, комплексную систему, которую невозможно воплотить в жизнь, не введя в процесс разработки определённый уровень абстракции. MVC позволяет легко превратить тысячи строк кода в стройную и логично устроенную систему, которую легко можно объять взглядом одного разработчика, и лишает разработку и последующее развитие кода множества проблем по интеграции нового кода со старым.

В настоящее время паттерн Model-View-Controller широко используется в php-фреймворках (Symfony, CodeIgniter, Kohana, Yii и тд), а также в ASP.NET, Django, Ruby On Rails и множестве других разработок. Сегодня MVC можно назвать стандартом де-факто в веб-разработке, как, например, использование реляционных баз данных – они подходят не для всех задач, но для большинства; а в случаях, когда они не подходят, использование нестандартных решений порой вызывает больше проблем (например, с обучением специалистов), нежели не совсем подходящие, но более привычные средства.

Литература

1. http://ru.wikipedia.org/wiki/Шаблон_проектирования
2. http://msdn.microsoft.com/en-us/library/ff649643.aspx
3. http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html
4. http://www.math.rsu.ru/smalltalk/gui/mvc-rus.pdf

УДК 004.75

РАСПРЕДЕЛЕННЫЙ ТЕСТИРУЮЩИЙ КЛИЕНТ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ ПРОВЕДЕНИЯ СОРЕВНОВАНИЙ ПО ПРОГРАММИРОВАНИЮ[1]

Визовитин Николай Валерьевич, магистрант, Новосибирский Государственный Университет, Россия, Новосибирск, vizovitin@gmail.com

Введение

Многие современные соревнования по программированию (например, ACM ICPC) для оценки решений участников используют ту или иную специализированную тестирующую систему. Как правило, такие системы позволяют исполнять программу, написанную участником, в автоматическом режиме на заранее определенном наборе сценариев. В простейшем случае программа запускается на серии предварительно подготовленных тестов. Кроме того, осуществляется контроль над используемыми программой ресурсами, а также обеспечивается изоляция и равноправие при исполнении различных решений. Примером таких систем могут служить ejudge, PCMS2, PC^2, NSUts [1].

Большинство автоматических систем тестирования имеют клиент-серверную архитектуру. В рамках такого разделения сервер отвечает за взаимодействие с пользователями и принятие решений участников на проверку. Но непосредственно проверка корректности решений осуществляется на нескольких тестирующих клиентах. Участники не имеют прямого доступа к тестирующим клиентам. Как правило, тестирующие клиенты запускаются на отдельных физических машинах, равно как и сервер. Основная причина

[1] Работа выполнена при проведении НИР в рамках реализации ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009 – 2013 годы, ГК № П262 от 23.07.2009.

такого разделения – ресурсоемкость задачи тестирования. Такая архитектура также понижает связность компонентов системы и позволяет изолировать сбои отдельных клиентов. В случае, если один из клиентов вышел из строя, остальные продолжат свою работу и имеющиеся работы по проверке будут перераспределены на оставшиеся клиенты.

Обычно сеанс проверки тестирующим клиентом одного решения включает в себя получение исходного текста решения с сервера, его компиляцию, запуск на серии тестов, сверку результатов с эталонными и отправку результатов проверки на сервер. Запуск скомпилированного решения (как и некоторые другие задачи) производится в специализированной изолирующей среде, гарантирующей безопасное исполнение произвольного кода и накладывающей определенные решаемой участником задачей ограничения на доступные ресурсы.

Большинство тестирующих систем используют свою собственную реализацию тестирующего клиента, удовлетворяющую нуждам проведения отдельных соревнований и тренировок. Несмотря на то, что наблюдается движение в сторону свободного программного обеспечения, многие соревнования по программированию по-прежнему проводятся на платформе Microsoft Windows. Настоящая работа описывает особенности дизайна и реализации нового тестирующего клиента, обладающего рядом уникальных свойств и архитектурой, ориентированной на высокую отказоустойчивость, расширяемость и независимость от используемого сервера.

1. Причины создания распределенного тестирующего клиента

Многие из используемых ныне тестирующих клиентов устроены достаточно просто. Каждый клиент не зависит от других и общается напрямую с сервером, последовательно выполняя получаемые задачи. Изолирующая среда [2], как правило, использует лишь средства операционной системы для ограничения ресурсов, доступных изолируемому процессу. Такие клиенты обычно запускаются на отдельных физических машинах одинаковой конфигурации. При этом решается одна достаточно узкая задача – проверка решений участников в рамках одной или небольшого количества моделей запуска и оценки. При возникновении необходимости сильного изменения модели запуска решений (например, для обеспечения поддержки интерактивных или игровых задач с несколькими участниками), модели оценки суммарного результата или другого аспекта тестирующего клиента, необходимо соответствующим образом модифицировать сам тестирующий клиент и заново развернуть его на целевых машинах. В случае отказа какого-либо из тестирующих клиентов серверу необходимо будет опознать эту ситуацию и перераспределить задачи этого клиента на другие тестирующие клиенты. К сожалению, обычно нет возможности восстановить работу тестирующего клиента в автоматическом режиме, поэтому часто это делают вручную. Перечисленные особенности и недостатки характерны для многих тестирующих клиентов, в том числе и для тестирующего клиента, используемого в качестве основного, в олимпиадной тестирующей системе Новосибирского государственного университета NSUts.

Для решения этих и других проблем ведется разработка нового тестирующего клиента. Новый клиент является распределенным – он также может содержать несколько тестирующих узлов, но они являются подчиненными другим узлам или основному узлу. Основной узел фактически является единственным, непосредственно взаимодействующим с сервером. Он получает задачи и распределяет их между подконтрольными ему узлами, а также следит за их работоспособностью. Это позволяет, с одной стороны, полностью сосредоточить логику, относящуюся к тестированию, в тестирующем клиенте. Серверу при этом достаточно знать лишь то, каким образом послать задачу на проверку и как передать тестирующему клиенту данные. С другой стороны, такой подход дает возможность централизованно управлять и следить за процессом тестирования, вне зависимости от используемой тестирующей системы. Также такая архитектура позволяет ввести единую систему мониторинга работоспособности отдельных узлов, не зависящую от тестирующей

системы. Отметим, что независимость от тестирующей системы не означает невозможность интеграции соответствующих данных и управляющих элементов в ее интерфейс.

2. Изолирующие среды

Одной из основных задач, решаемых новым тестирующим клиентом, является поддержка нескольких изолирующих сред. Классический подход к построению изолирующей среды посредством использования стандартных средств ОС для ограничения доступа к ресурсам показал свою уязвимость к ряду целенаправленных атак [3]. К таким атакам, в частности, относятся атаки, направленные на ядро ОС [4]. Поэтому в ходе работы была создана изолирующая среда на основе технологий виртуализации. В этой изолирующей среде используется виртуальная машина Oracle VirtualBox [5] с оптимизированным настройками и специальным образом подготовленный минимальный образ гостевой ОС на основе Windows XP. В образе гостевой ОС были оставлены лишь самые необходимые для проведения на нем тестирования компоненты. В результате была получена изолирующая среда с приемлемыми ресурсными издержками, обеспечивающая высокий уровень безопасности и изоляции. Использование изолирующей среды на основе технологий виртуализации имеет и другие существенные преимущества, например, возможность запуска более одного экземпляра изолирующей среды на одной машине.

Новый тестирующий клиент поддерживает использование нескольких изолирующих сред, а также, в некоторых случаях, вложенное их исполнение. Так, поддерживается и изолирующая среда на основе технологий виртуализации, и реализация классической изолирующей среды для Windows [2]. Отметим, что первая использует облегченную версию второй для более точного контроля за доступными приложению ресурсами. Новые типы изолирующих сред могут быть добавлены к тестирующему клиенту в виде модулей. Поддержка нескольких изолирующих сред и изолирующая среда на основе технологий виртуализации являются уникальными особенностями среди публично доступных тестирующих клиентов.

Дополнительные модели запуска решений и их оценки также могут быть добавлены в виде модулей тестирующего клиента. Так, например, может быть задана «игровая» модель запуска, в которой одновременно запускаются два или более решений различных участников и связываются между собой и управляющей программой для организации игры. На данный момент реализовано две модели оценки решений – по системе ACM и по школьной системе. Первая подразумевает исполнение решения до первой ошибки. В рамках второй каждому тесту сопоставляется некоторое количество баллов, баллы за правильно пройденные тесты суммируются.

3. Организация взаимодействия между компонентами системы

Протокол взаимодействия клиента и сервера версионирован и расширяем. Единица передачи данных в рамках этого протокола – сообщение. Сообщения, за небольшим количеством исключений, являются документами XML. В качестве транспортного протокола используется HTTP, однако транспортный уровень может быть заменен. Так, его легко заменить протоколом TCP или HTTPS (например, с целью обеспечения безопасной передачи данных). Несмотря на использование на транспортном уровне протокола, подразумевающего взаимодействие типа запрос-ответ, передача и обработка сообщений асинхронна – после передачи сообщения источник не дожидается ответа приемника. Протокол имеет декларативный характер, в отличие от более распространенных протоколов вида «удаленный вызов процедур» (RPC) [6]. Таким образом, он ориентирован в первую очередь на передачу данных и исполнение запросов, а не на выполнение команд. Для передачи и описания данных в сообщениях используются так называемые артефакты. Артефакт имеет тип, идентификатор, полезную нагрузку в виде интерпретируемых данных, набор зависимостей от других артефактов (отражающих зависимости по данным), и метаданные, указывающие, как следует интерпретировать переносимые артефактом данные.

Для организации взаимодействия между узлами тестирующего клиента используется аналогичный протокол. В качестве транспортного протокола в этом случае используется TCP для оптимизации быстродействия посредством исключения многократных установок соединения, присущих HTTP. Тестирующие узлы взаимодействуют друг с другом в соответствии со своими положениями в общей иерархии, однако при посредничестве родительского узла может быть установлена прямая связь между двумя узлами из различных поддеревьев. Это позволяет уменьшить издержки во время передачи больших объемов данных, например в случае, если результаты работы решения участника должны быть отправлены на другой узел с целью проверки. Такая ситуация может иметь место при специализации узлов по выполняемым задачам (например, компиляция, исполнение и проверка), что также возможно только при использовании распределенного тестирующего клиента.

4. Особенности реализации тестирующего клиента

В качестве основного языка для реализации тестирующего клиента был выбран Python, а именно его диалект, реализуемый интерпретатором Stackless Python [7]. Python является достаточно простым и элегантным языком с богатой стандартной библиотекой. Он также является мультиплатформенным, и его использование облегчает одновременную разработку под операционные системы семейства Windows и Linux. Так как в некоторых сценариях тестирующий клиент должен асинхронно обрабатывать большие объемы данных, выбор пал на Stackless Python, как реализацию, предоставляющую эффективные средства выполнения этой задачи. Основная особенность этого интерпретатора заключается в улучшенной поддержке многопоточности.

Развертывание типовой сборки тестирующего узла уже сейчас является достаточно простой задачей, для выполнения которой не нужен специалист. Однако при изменении кода тестирующих узлов по-прежнему необходимо их повторное развертывание. Планируется заменить установку полноценного тестирующего узла установкой лишь базового агента, который, получив необходимые компоненты от основного узла, сможет стать полноценным тестирующим узлом без вмешательства пользователя. Аналогичным образом обновления программного обеспечения узлов могут быть применены централизованно с основного узла.

Новый тестирующий клиент включает систему мониторинга работоспособности и загруженности тестирующих узлов. Эта подсистема является внешней по отношению к тестирующим узлам и состоит из агентов на физических машинах, где работают контролируемые объекты, и системы опроса состояния узлов. При существенном отклонении от штатного режима функционирования (например, тестирующий узел не отвечает на запросы) система мониторинга посылает сигнал соответствующему агенту, который завершает все процессы, порожденные тестирующим узлом, и перезапускает его. Администратор тестирующей системы при этом уведомляется об отклонениях в работе.

Выводы

В результате работы над новым тестирующим клиентом была создана изолирующая среда на основе технологий виртуализации и прототип клиента. Прототип использовался как вторичный тестирующий клиент в системе NSUts, а также подвергался синтетическим тестам. Новый клиент показал хорошие результаты, как по производительности, так и по стабильности работы. Планируется его апробация на нескольких соревнованиях по программированию.

После достижения необходимого уровня качества разрабатываемый тестирующий клиент будет опубликован и доступен для использования в других тестирующих системах. Использование такого клиента позволит разработчикам тестирующих систем, с одной стороны, сфокусироваться на реализации необходимой функциональности и бизнес-логики системы. И, с другой стороны, даст платформу для создания тестирующего клиента, удовлетворяющего определенным нуждам. Кроме того, несмотря на достаточно узкую решаемую задачу – запуск недоверенного кода, исполняющегося короткое время, с

последующей оценкой результатов – разрабатываемый тестирующий клиент может быть использован не только на соревнованиях по программированию, но и, например, для организации сетевого сервиса по удаленному исполнению кода.

Литература

1. Боженкова Е.Н., Иртегов Д.В., Киров А.В., Нестеренко Т.В., Чурина Т.Г. Автоматизированная система тестирования NSUts: Требования и разработка прототипа // Вестник НГУ, серия: Информационные технологии. – N4, Т.8, 2010. – С. 46-53.
2. Киров А.В. Изолирующая среда для запуска тестовых прикладных программ // Материалы VII Всероссийской научно-практической конференции студентов, аспирантов и молодых ученых «Студент и современные информационные технологии». – Томск: Томский политехн. ун-т. 2009. – С. 285-286.
3. David Maynor. Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research – Syngress, 2007 – 350 p.
4. Greg Hoglund, Jamie Butler. Rootkits: Subverting the Windows Kernel – 1st Edition – Addison Wesley, 2005 – 352 p.
5. Официальный сайт виртуализирующего ПО Oracle VirtualBox. [Электронный ресурс]. URL: http://virtualbox.org/
6. Joe Johnston, Edd Dumbill, Simon St.Laurent, John Posner. Programming Web Services with XML-RPC – 1st Edition – O'Reilly Media, 2001 – 230 p.
7. Официальный сайт интерпретатора Stackless Python. [Электронный ресурс]. URL: http://www.stackless.com/

УДК 519.682.4+ 371.32

ОБУЧЕНИЕ СТАНДАРТНОЙ БИБЛИОТЕКЕ ШАБЛОНОВ STL ЯЗЫКА C++ НА ПРИМЕРЕ РАЗРАБОТКИ КОМПРЕССОРА ПО МЕТОДУ ХАФФМАНА

Штанюк Антон Александрович, к.т.н., доцент, Нижегородский коммерческий институт, Россия, Нижний Новгород, shtan@land.ru

Стандартная библиотека шаблонов языка C++ является мощным средством обобщённого и объектно-ориентированного программирования, значительно упрощающим процесс разработки крупных программных систем. Обучение данному средству очень важно в современных курсах подготовки специалистов в области информационных систем. Однако часто преподаватели испытывают недостаток в реальных примерах, на которых бы иллюстрировалось использование элементов STL. В рамках настоящей работы предлагается такой пример, имеющий практическое значение: программа-компрессор, сжимающая отдельные файлы.

Компрессор по методу Хаффмана [1] относится к классу программ, сжимающих файлы за счёт уменьшения избыточности, вызванной использованием равномерного ASCII-кодирования при наличии разных вероятностей появления элементов алфавита. Несмотря на то, что строковое сжатие более эффективно в практическом отношении, символьное сжатие хорошо иллюстрирует идеи теории информации, а также требует использования множества конструкций языка программирования при реализации, позволяя достичь целей обучения. В настоящей работе речь пойдёт об использовании стандартной библиотеки шаблонов при разработке компрессора. Данный пример хорошо подходит для иллюстрации возможностей объектного подхода при решении задач с привлечением контейнеров и алгоритмов STL [2] и может быть рекомендован в качестве учебного примера в рамках курсов по объектно-ориентированному программированию на C++.

Компрессор принимает на вход обрабатываемый файл (1) и анализирует его. Результатом анализа выступает частотная таблица символов (байт) исходного файла (2). На основе этой таблицы генератор кода строит бинарное дерево Хафманна, необходимое для получения префиксного кода (3), который наилучшим образом подходит для представления

информации в файле. Кодер читает входной файл снова и перекодирует его в соответствии с новым префиксным неравномерным кодом (4), выдавая закодированное представление в виде текстовой строки из нулей и единиц. Закодированное содержимое подвергается упаковке (осуществляется перевод из строкового представления в битовое) и помещается в результирующий (сжатый) файл вместе с заголовком, содержащим служебную информацию.

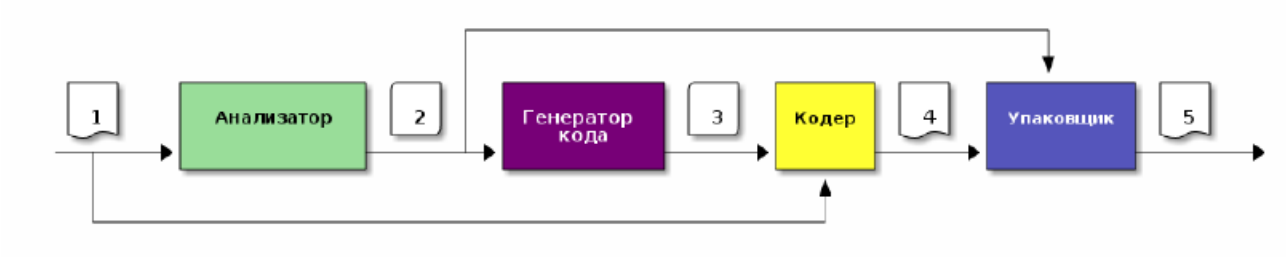


Рис. 1 – Схема компрессора

Опишем кратко те части программы, в которых используются элементы STL. Основной структурой для хранения информации об одном элементе ASCII-алфавита является структура SYM:

```
struct SYM
{  unsigned char ascii; // ASCII-код (исходный) символа
   float freq;          // частота встречаемости в исходном файле
   string code;         // новый код по Хаффману
   SYM *left;           // для построения дерева
   SYM *right;          // для построения дерева
};
```

Структура содержит два указателя на SYM, позволяющих связывать объекты друг с другом при построении бинарного дерева.

Для хранения информации обо всех символах алфавита мы применяем контейнер vector из STL. Для доступа ко всем элементам вектора создаётся соответствующий итератор

```
vector<SYM*> psym;
vector<SYM*>::iterator psymIter;
```

Для того, чтобы сформировать вектор, нам понадобится вспомогательный словарь symStat, являющийся контейнером STL map

```
map<unsigned char, long> symStat;
map<unsigned char, long>::iterator symIter;
```

Словарь symStat мы используем для сбора статистики встречаемости символов исходного файла. Он представляет собой пары «символ – счётчик» и легко заполняется при движении по файлу. После сканирования исходного файла мы имеем информацию о количествах всех символов и на её основе формируем вектор указателей psym.

Каждый элемент вектора psym представляет собой адрес структуры SYM, описывающий один уникальный символ исходного файла. Рассмотрим алгоритм построения бинарного дерева Хаффмана для файла, содержащего последовательность символов:

```
zzxz.yzyxxzxy.yzxxxzy.yx.
zxyyxzyx.xyzyxxzzyyz.xzxy
.xxzxyxxzxyyz.yxxzxyyzyxx
zx.zzxyyyxzxxxyxx.zxxxzxx
```

Слева приведены записи о символах исходного файла, представленные в виде массива и отсортированные в порядке убывания частоты. На основе двух символов с наименьшей частотой мы создаём узлы дерева и снова включаем их в массив. Процесс рекурсивно повторяется до тех пор, пока в массиве не окажется единственный элемент, выступающий в роли корня дерева.

Рассмотрим реализацию функции построения дерева. В ней используется контейнер vector, содержащий указатели на узлы (SYM), а также вызывается алгоритм sort для сохранения порядка следования элементов в векторе. Для обработки содержимого вектора

вызываются методы push_back() и pop_back() для помещения и удаления элементов, соответственно.

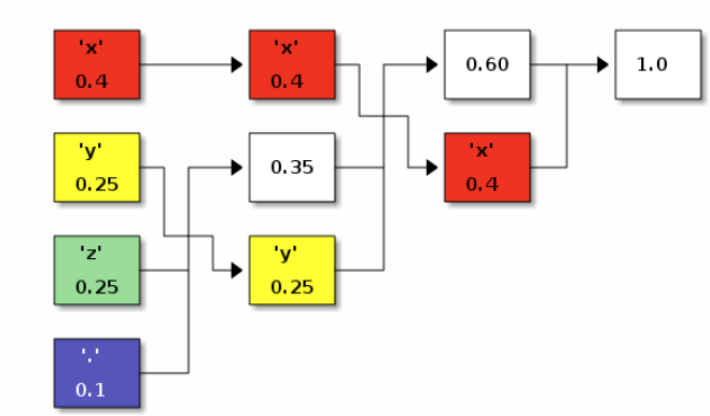


Рис. 2 – Схема построения дерева Хаффмана

```
bool compare (SYM *a,SYM *b) { return (a->freq>b->freq); }
SYM *makeHuffTree(vector<SYM*> &psym)
{     // создаём новый элемент
  SYM *temp = new SYM;
  temp->freq = psym[psym.size()-1]->freq+psym[psym.size()-2]->freq;
  // образуем связи для построения дерева
  temp->left = psym[psym.size()-1];
  temp->right = psym[psym.size()-2];
  // дерево построено, если в массиве остались 2 элемента
  if(psym.size()== 2)
        return temp;
  // помещаем новый элемент в массив
      psym.push_back(temp);
  // сортируем массив по убыванию частот
      sort(psym.begin(),psym.end(),compare);
  // удаляем из массива 2 последних элемента
      psym.pop_back();
      psym.pop_back();
  // рекурсивно вызываем себя для продолжения построения дерева
  return makeHuffTree(psym);
}
```

На следующем этапе мы выполняем рекурсивный обход дерева, с добавлением в поле code каждого узла 1 и 0, в зависимости от перехода в левое или в правое поддерево.

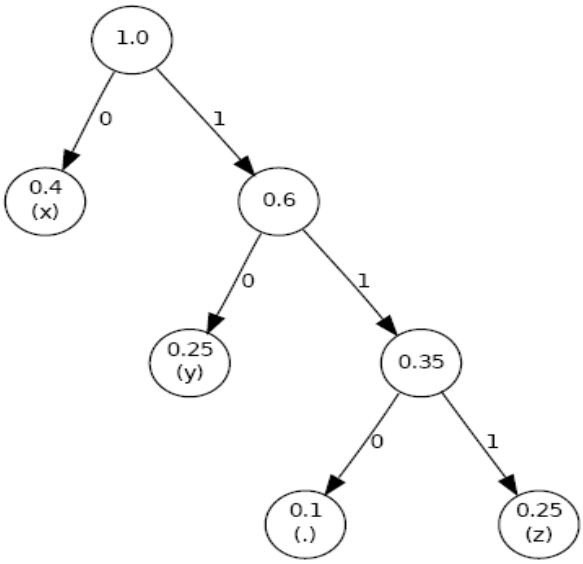


Рис. 3 – Дерево Хаффмана и схема кодирования

Для ускорения процесса кодирования исходного файла создаётся специальный словарь symCodes с исходными символами и кодовыми комбинациями на основе рекурсивного обхода построенного дерева

```
void makeCodesMap(SYM *root, map<unsigned char,string>& symCodes)
{
   if(root->left)
      makeCodesMap(root->left,symCodes);

   if(!root->left && !root->right)
      symCodes[root->ascii]=root->code;

   if(root->right)
      makeCodesMap(root->right,symCodes);
}
```

На этапе кодирования происходит формирование строкового потока с закодированным представлением содержимого исходного файла. Для каждого символа из обрабатываемого файла определяется кодовая комбинация из словаря, а затем эта комбинация поступает в строковый поток.

```
// строковый поток для кодовой последовательности
stringstream data101 (stringstream::in | stringstream::out);
// цикл чтения входного файла (in) для кодирования методом Хаффмана
while(1) {
   ch=in.get();
   if(ch==-1)
     break;
   else {
      // накапливаем закодированные данные
     data101<<symCodes[(unsigned char)ch];
     codelen+=symCodes[(unsigned char)ch].length();
   }
}
```

Как только закончено формирование закодированного представления исходного файла, необходимо произвести его сжатие. Для решения этой задачи используется контейнер bitset, который заполняется на основе строки из восьми нулей и единиц. Благодаря специальному методу to_ulong() удаётся получить сжатое представление данных и вывести его в результирующий файл вместе с заголовком.

```
while(!data101.eof()) {
 data101.read(buf,8);
 string temp=buf;
 bitset <8> eightbits (temp);
 out.put((unsigned char)eightbits.to_ulong());
 memset(buf,0,8);
}
```

Таким образом, данный учебный пример показывает применение таких элементов STL, как контейнеры vector, bitset, map, итераторов для доступа к элементам контейнеров, алгоритма sort и строковых, файловых и стандартных потоков ввода/вывода.

Литература

1. Д. Сэломон. Сжатие данных, изображений и звука. – М: Техносфера, 2006.
2. Лишнер Р. STL. Карманный справочник. – СПб.: Питер, 2005.

УДК 004.89

АГЕНТНО-ОРИЕНТИРОВАННАЯ ТЕХНОЛОГИЯ РАЗРАБОТКИ РАСПРЕДЕЛЕННЫХ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Зайцев Евгений Игоревич, к.т.н., доцент, Московский государственный университет приборостроения и информатики, Россия, Сергиев Посад-7, zei@tsinet.ru

Распределенные интеллектуальные системы могут быть построены на базе обычных совместно функционирующих объектов, при этом ориентация на коллективный потенциал, а не на индивидуальные возможности объектов, не превращает объектно-ориентированную систему в агентно-ориентированную. Внимание к агентно-ориентированной технологии инвесторов, а также отсутствие общепринятого формального определения программного агента приводят к тому, что многие разрабатываемые лабораториями, университетами, фирмами и промышленными организациями объектно-ориентированные системы позиционируются как многоагентные (MAC, MAS–MultiAgent Systems) [1-5].

Агентно-ориентированный подход, подобно ориентированной на службы архитектуре SOA (Service-Oriented Architecture), предполагает разбиение приложений на крупно гранулированные, слабо связные компоненты, которые инкапсулируют больше функциональности, по сравнению с такими программными абстракциями, как методы и классы. Теория агентов и многоагентных систем предлагает такие высокоуровневые понятия как роли агентов, планы, цели, протоколы общения и ведения переговоров. В отличие от традиционных объектно-ориентированных программных серверов, имеющих развитые средства взаимодействия со средой и другими объектами и предоставляющих определенные услуги клиентам, агенты способны действовать рационально и осуществлять логические выводы в условиях неполноты и противоречивости получаемой информации.

С позиций объектно-ориентированного программирования программный агент можно рассматривать как особый тип объекта (smart object), который содержит когнитивные структуры данных (cognitive data sructures) и методы, реализующие достоверный (дедуктивный) или правдоподобный (индуктивный, абдуктивный) выводы. Кроме того, модель агента интегрирует механизмы рассуждения на основе знаний с нейросетевым (коннекционистским) подходом, смещающим акцент исследований с проблем символьного представления и организации формальных рассуждений на проблемы обучения и адаптации. Структура коннекционистских систем формируется не только при их разработке, но и в процессе взаимодействия со средой.

Особенность агентов состоит в том, что в их модели присутствуют ментальные свойства (интенциональные характеристики): убеждения, желания и намерения (модель BDI, Belief, Desires, Intentions – минимальное интенциональное описание агента), которые направляют деятельность агентов и могут меняться в процессе функционирования. При формализации желаний агента, можно сказать, что они представляют собой состояния (ситуации), достижение которых важно для агента. Желания можно свести к убеждениям относительно будущих действий, при этом, в противоположность убеждениям (истинным или ложным), желания рассматриваются не с позиций их истинностного значения, а в контексте удовлетворения или выполнения. К исполнению желания агента ведёт реализация его намерений – того, что агент должен (собирается) сделать в силу своих обязательств и/или желаний. Для удовлетворения желаний агент может, не меняя своих намерений и убеждений в том, что действия, которые он намерен совершить, возможны, делать попытки соответствующим образом воздействовать на внешнее окружение.

Не обладая полным знанием о своем окружении и имея лишь частичное представление о проблеме, агенты проводят неточные, предположительные, правдоподобные рассуждения. Эти рассуждения подвергаются пересмотру (belief revision) при получении агентом дополнительной информации, несовместимой с полученными ранее заключениями, а также при изменении модели мира в результате обновления убеждений агентов (belief update).

Убеждения агента становятся знанием, если их истинность подтверждается (обосновывается) новыми фактами или объяснениями. Немонотонные логики, такие как логика умолчания (default logic), автоэпистемическая (autoepistemic logic), немонотонная модальная логика (modal nonmonotonic logic), косвенное описание (circumscription), позволяют решить проблему отсутствия у агентов определенных знаний, при этом нет необходимости придерживаться предположения о замкнутом мире (closed-world assumption), согласно которому высказывания, истинность которых не может быть доказана, считаются ложными, а имеющаяся в базе знаний информация полной [6].

Кроме логических выводов в условиях неопределенности агенты осуществляют мониторинг событий, выполнение триггерных и других операций в объектно-реляционных базах данных. При обработке четких SQL-запросов часто встречается ситуация когда либо сложно получить точно соответствующий запросу ответ, либо в результате запроса будет получен слишком большой объем несущественных для решения задачи данных. Используемые агентами нечеткие запросы (fuzzy queries) позволяют расширить область поиска в соответствии с изначально заданными ограничениями. Применяя нечеткие модификаторы и задавая соответствующие диапазоны числовых величин, можно получить информацию, в соответствующем приближении, удовлетворяющую заданным критериям. Для реализации нечетких запросов строятся нечеткие отношения с использованием соответствующих функций принадлежности, на основе которых формулируются и реализуются запросы в нечетком виде.

Разбиение приложения на автономные, крупно гранулированные служебные компоненты облегчает идентификацию естественного параллелизма, который существует в контексте предметной области, и понимание того, как следует проводить декомпозицию работ, которые можно выполнять одновременно. Разработчики, реализующие распределенные интеллектуальные системы на основе агентов, могут сосредоточиться на корректном моделировании задач, выполняемых агентами, а не на стремлении явно управлять параллелизмом в программе. Повышение уровня используемых абстракций упрощает разработку распределенных интеллектуальных систем, при этом оно ограничивает область действия абстракций и объем контроля деталей реализации, порученного разработчикам. Повторное использование абстракций высшего уровня сопряжено с трудностями. Чтобы быть многократно используемой, абстракция должна допускать адаптацию, как через некоторые внутренние механизмы изменчивости, так и через внешние адаптеры. Фундаментом для систематического повторного использования в настоящее время служат проблемно-ориентированные языки, определенные множеством классов каркасы, адаптация которых возможна только в точках расширения (в абстрактных классах, для которых не приведены реализации), а также шаблоны (patterns), отличающиеся от каркасов тем, что они не предусматривают специфической реализации. Для разработки прикладных МАС удобно использовать управляемые моделями среды разработки, которые позволяют упростить процесс создания экземпляров и сборки агентов, обеспечивают автоматическую генерацию частичных или полных реализаций на основе спецификации [7].

Для поддержки процессов проектирования и реализации многоагентных систем существует достаточно большое количество инструментальных средств, которые формируют среду, оптимизированную для выпуска определенного типа приложений. В настоящее время используют инструментальные средства, которые, в большинстве случаев, основаны на Java и фокусируются либо на реализации спецификаций стандарта Международного Фонда интеллектуальных физических агентов (Foundation for Intelligent Physical Agents, FIPA), либо спецификаций стандарта MASIF ассоциации OMG (Object Management Group) по реализации систем мобильных агентов и обеспечению интероперабельности между различными архитектурами. К первой группе относятся такие инструментальных средства, как JADE (Telecom Italia Lab), Agent Factory (PRISM Laboratory), FIPA-OS (Emorphia), Jack (AOS Group), Zeus (British Telecommunication), Agent Development Kit (Tryllian BV), Grasshopper (IKV technologies AG). Вторую группу представляют такие среды, как Aglets SDK (IBM),

Odyssey (GenMagic), D'Agents (Dartmouth college), а также Grasshopper. Многие из этих систем позиционируются, как проекты с открытым исходным кодом (JADE, ZEUS, FIPA-OS, AgentFactory, Tryllian ADK).

Фонд FIPA унифицирует архитектуру платформ агентов, необходимые для управления агентами операции и способы взаимодействия агентов с программным обеспечением, которое не использует агентную технологию [8]. Согласно FIPA система агентов содержит подсистему управления агентами (Agent Management System), службу каталога (Directory Facilitator), канал связи агентов (Agent Communication Channel) и менеджер безопасности платформы агентов (Agent Platform Security Manager). Данные компоненты включают сервисы по созданию, удалению, деактивации, возобновлению работы и миграции агентов, службы маршрутизации сообщений и управления жизненным циклом, службы белых страниц (с информацией о соответствии между глобально уникальными именами агентов и адресами локального транспорта, используемого платформой) и желтых страниц, службы каталога (с описанием агентов и услуг, которые они предоставляют), а также сервисы по осуществлению политики безопасности на транспортном уровне.

Стандарт OMG MASIF (Mobile Agent System Interoperability Facilities), представляя собой надстройку над стандартом распределенного кроссплатформенного объектно-ориентированного программирования CORBA (Common Object Request Broker Architecture), позволяет объединить традиционную клиент-серверную парадигму и технологию мобильных агентов. Аналогично FIPA, стандарт MASIF унифицирует синтаксис и правила выполнения операций, связанных с созданием, удалением, перемещением и идентификацией агентов, приостановкой и возобновлением их работы, получением агентом информации о типе платформы. Поверх брокеров объектных запросов (ORB) стандарта CORBA могут применяться коммуникационные протоколы прикладного уровня, называемые языками взаимодействия агентов (ACL, Agent Communication Language).

В случаях, когда узлы должны динамически перестраиваться на решение различных задач, используются мобильные агенты, поддерживающие либо слабую (weak mobility), либо сильную (strong mobility) модели мобильности. Интеллектуальные мобильные агенты обычно требуют поддержки сильной мобильности, при которой вместе с сегментом кода переносится также сегмент исполнения, что позволяет работающему процессу после приостановки и перенесения на другую машину продолжить его выполнение с того места, на котором этот процесс был приостановлен. Примером агентов с сильной мобильностью являются Java-аглеты (aglets), которые пересылаются из одного контекста в другой по протоколу ATP (Agent Transfer Protocol) прикладного уровня, не зависящему от платформы и использующему унифицированный указатель ресурса (URL, Uniform Resource Locator) для определения местоположения агентов и серверов.

Разработка языков и инструментов для создания прикладных МАС возможна с использованием средств расширяемости, предоставляемых современными интегрированными средами разработки (Integrated Development Environment, IDE). Разработка подключаемых модулей специального назначения для существующих IDE, например, таких как мультиплатформенные Eclipse (Eclipse Foundation) или NetBeans (NetBeans Community), облегчает построение специализированных инструментальных средств, нацеленных на проектирование распределенных интеллектуальных систем, и обеспечивает высокие уровни специфичности и автоматизации по относительно невысокой цене. Поскольку Eclipse и NetBeans состоят из плагинов (PDE), у разработчиков инструментальных средств имеется возможность предложить свои расширения к этим IDE и предоставить пользователям последовательную и цельную интегрированную среду разработки. Хотя Eclipse и NetBeans написаны на Java, использовать их можно и с другими языками. Более того, используя обычные приемы программирования, можно собрать язык моделирования из компонентов, которые входят в библиотеку компонентов определения языков в IDE.

Высокий уровень абстракции, используемый при агентно-ориентированном подходе, и выросшая мощность платформенных технологий (от библиотек классов к каркасам компонентов, промежуточному программному обеспечению на основе распределенных компонентов, и асинхронной слабо связанной архитектуре служб) позволяют использовать концепцию программных агентов при разработке распределенных интеллектуальных систем учебного назначения и, в частности, многоагентных банков знаний (МБЗ) [9]. МБЗ представляют собой распределенные интеллектуальные информационные системы, которые формируют ответы на запросы пользователей посредством выполнения специализированных процедур поиска и логической обработки знаний, а также выступают в качестве адаптивных обучающих систем.

Многоагентный банк знаний включает общие и специальные знания о предметной области, о процессе обучения и модели обучаемого, ассоциируя их с реактивными и когнитивными программными агентами [10-11], которые реализуют процедуры обработки этих знаний, выдают ответы на запросы пользователей и строят рациональную стратегию обучения, совершенствующуюся по мере накопления данных. Для формирования ответов на запросы пользователей о значениях различных характеристик объектов и событий, о сравнении и анализе событий, выявлении связей между событиями, а также запросы о формировании упорядоченных совокупностей событий, обеспечивающих решения тех или иных задач, в МБЗ вместо одного интеллектуального решателя используется многоуровневая сеть программных агентов.

Использование для формирования ответов на запросы пользователей одновременно нескольких программных агентов не только повышает производительность системы за счет параллелизма, но и расширяет возможности банков знаний по предоставлению пользователям обобщенной информации. Агенты, распределенные в узлах локальной или глобальной вычислительной сети, способны предоставлять или рекомендовать обучающие материалы на основе обобщения предпочтений, поведения и представлений определенных групп пользователей системы.

Ответы на запросы пользователей формируются когнитивными агентами МБЗ в результате спецификации свойств сущностей (событий и их субъектов), вычисления каузальных, временных и других отношений на множестве сущностей, а также в результате планирования решения задач. При этом вычисление отношений и синтез плана действий для решения некоторой задачи происходят не только в процессе выполнения когнитивным агентом продукционных, редукционных или трансформационных правил, но также в процессе его переговоров с другими агентами. Многоуровневая архитектура многоагентного банка знаний предполагает использование как горизонтальных, так и вертикальных связей между агентами. В ней присутствуют уровни, ответственные за кооперативное поведение, локальное планирование, формирование намерений, восприятие и исполнение действий, реактивное поведение и обучение агента. При этом каждый агент функционирует в соответствии с кооперативными обязательствами, которые возложены на агента другими агентами МБЗ.

В двухуровневой архитектуре многоагентного банка знаний модель реактивного агента может быть задана следующим образом:

$A_R=(Z_R,\ W_R,\ N(Z_R,Net,S_R),\ S_R(R,A(G)))$, где Z_R – множество входных сообщений; W_R – множество выходных сообщений; N – множество методов, определяющих реакции нейронной сети *Net* реактивного агента на входные сообщения Z_R; S_R – множество состояний, каждое из которых определяется набором атрибутов агента и их значениями:

$INT\ R_i = \{...[A_j\ ,DOM(A_j)],...\};\ EXT\ R_i = \{F_1,...F_p\ \};\ F_k=\{A_1(G_1),...\ A_s(G_n)\}$, где R – множество отношений, G – множество значений множества атрибутов A. Домены (*DOM*) являются общими совокупностями значений, из которых берутся реальные значения для атрибутов отношения. Интенсиональные части *(INT)* локальных баз данных содержат информацию, характеризующую семантику предметной области, экстенсиональные части *(EXT)* описывают возможные состояния агентов и их взаимосвязи. Одноуровневая

многоагентная система, состоящая из самоорганизующихся реактивных агентов, не обладает центральным планом действий, имеет апостериори определяемые организационные структуры и формируется путём простой локальной адаптации. Адекватные реакции агентов на возможные в окружающей среде ситуации реализуются благодаря обучению нейронной сети *Net*, которая поддерживает механизм конкуренции между желаниями и позволяет при предъявлении агенту входного вектора (ситуации) возбуждать наиболее точно соответствующие ему действия.

В случае многоуровневой организации МБЗ помимо реактивных агентов, не имеющих целевой ориентации и связанных с набором источников знаний о предметной области, функционируют когнитивные агенты, которые планируют собственные действия и координируют деятельность реактивных агентов. Модель когнитивного агента формально может быть определена шестеркой: $A_K=(Z_K,W_K,S_K,,SP,P)$, где Z_K – множество входных сообщений; W_K – множество выходных сообщений (осведомительных, управляющих, координационных); S_K – множество состояний когнитивного агента, соответствующее множеству ситуаций, которые зависят от ментальных свойств агента (убеждений, намерений, желаний, выполняемых действий), состояний других агентов и их взаимосвязей, происходящих в системе событий; SP – система продукций, определяющая переходы агента из одного состояния в другое и формируемые при этом выходные сообщения, инициирующие синхронные либо асинхронные операции обмена; $P=(D,SG,s_0)$ – система планирования в пространстве состояний для проблемной области D с исходным состоянием s_0 и динамическим множеством целей SG.

Создание многоагентных банков знаний является сложной задачей, которая требует от разработчиков опыта проектирования и реализации как концептуальных, так и технических решений в таких областях, как представление и обработка знаний, сетевые коммуникации и протоколы взаимодействия, нейронные сети и нечеткая логика. Проектирование и реализация МБЗ значительно упрощается при использовании специальных инструментальных средств, поддерживающих процесс создания экземпляров и сборки агентов. Специализированные инструментальные средства разработки многоагентных систем, подобные разрабатываемой в Московском государственном университете приборостроения и информатики проблемно-ориентированной среде AgentITS, поддерживают процессы проектирования и реализации обучающих агентов с заданным поведением.

Среда AgentITS представляет собой специфичную для проблемной области RAD. Инструментарий AgentITS, состоящий из интерактивных мастеров и панелей свойств, формирует среду, оптимизированную для создания распределенных интеллектуальных систем учебного назначения, которые должны формировать ответы на запросы пользователей посредством выполнения специализированных процедур поиска и распределенной обработки знаний, а также осуществлять адаптивное обучение с использованием персональных обучающих агентов. Инструментарий AgentITS включает такие группы программных средств, как среда выполнения МАС (агентная платформа) и инструментальная среда разработки МБЗ.

Агентская платформа отвечает за обеспечение жизнедеятельности агентов в составе МАС и представляет собой систему промежуточного уровня (middleware), которая находится между агентами и операционной системой. Основные функции агентной платформы состоят в управлении агентами, обеспечении передачи сообщений между агентами, в поиске агентов и данных о них внутри системы, поддержке онтологий. Платформа агентов позволяет просто передавать, принимать, регистрировать агентов, обеспечивает безопасность узла и устойчивость, то есть способность агентов восстанавливать свое состояние после аварийного завершения. Инструментальная проблемно-ориентированная среда AgentITS упрощает процесс создания персональных интеллектуальных агентов, а также агентов обучающих ресурсов, осуществляющих доступ к учебным материалам с удаленных компьютеров.

Литература

1. Тарасов В.Б. От многоагентных систем к интеллектуальным организациям. – М.: Эдиториал УРСС, 2002. -352с.
2. Городецкий В.И., Грушинский М.С., Хабалов А.В. Многоагентные системы / Новости искусственного интеллекта, №2, 1998.
3. Городецкий В.И., Карсаев О.В., Конюший В.Г., Самойлов В.В., Хабалов А.В. Среда разработки многоагентных приложений MASDK // Информационные технологии и вычислительные системы, 2003, №12. С.26-41.
4. Pavon J. The INGENIAS Methodology and Tools / J. Pavon, J. Gomez-Sanz Ruben Fuentes // Agent-Oriented Methodologies, Brian Henderson-Sellers & Paolo Giorgini (eds.), Idea Group Publishing, 2005 – P. 236-276.
5. Weiss G. Multiagent systems. MIT Press, Cambridge, Massachusetts, 1999.
6. Рассел С., Норвиг П. Искусственный интеллект: современный подход. –М.: Вильямс, 2006. -1408с.
7. Greenfield J., Short K. Software Factories: assembling applications with patterns, models, frameworks and tools. Wiley Pablishing, Inc., 2004.
8. Bellifemine F., Poggi A., Rimassa G. Developing multi-agent systems with a FIPA-compliant agent framework // Software – Practice And Experience. – 2001, № 31(2) – P. 103-128
9. Зайцев Е.И. О концепции многоагентных банков знаний, как интеллектуальных обучающих системах // Материалы 16-й международной конференции “Современное образование: содержание, технологии, качество”. В 2-х.т. – СПпб.:Изд-во СПбГЭТУ “ЛЭТИ”. 2010. Т.2. С. 36 – 38.
10. Зайцев Е.И. Распределенная интеллектуальная система на базе программных агентов с нечёткими знаниями // Информационные технологии, №9, 2006.
11. Зайцев Е.И. Методология представления и обработки знаний в распределенных интеллектуальных информационных системах. // Автоматизация и современные технологии. 2008, №1, С.29-34.

УДК 004.054

МАТЕМАТИЧЕСКИЕ МОДЕЛИ ТЕСТИРУЕМОГО ИСХОДНОГО КОДА

Денисов Виктор Сергеевич, аспирант, Саратовский государственный университет им. Н. Г. Чернышевского, Россия, Саратов, denisovenator@gmail.com

Введение

В данной статье предлагаются ряд формальных понятий и математических моделей для оценки пригодности исходного кода к модульному тестированию в средах JUnit и TestNG. Модульное тестирование было выбрано для анализа, так как оно является важной составляющей современных гибких методик разработки[1]. Модульное тестирование поощряет изменение кода и рефакторинг[2], способствуя общему повышению качества исходного кода.

Вопросы пригодности исходного кода к модульному тестированию хорошо описаны в книге [3] в виде паттернов работы с унаследованным кодом. Однако книга в значительной мере рассчитана на применение описанных методов человеком и не содержит формальных понятий для оценки тестируемости исходного кода. В данной статье развивается идея объектного шва, как наиболее пригодного для сред тестирования Junit и TestNG. В результате были получены понятия внутреннего экземпляра и исходящей зависимости. На их основе был построен ряд математических моделей, которые позволяют оценить, насколько объектно-ориентированный код пригоден для модульного тестирования.

1. Используемые термины

Будем называть базовыми выражениями языка следующие выражения:

- выражение бинарного оператора;
- выражение унарного оператора;

- выражение присваивания;
- выражения приведения типов;
- выражение литерального значения (логического, строкового, числового);
- выражения вызова метода;
- выражение доступа к полю;
- выражение конструирования объекта;
- выражение доступа к массиву;
- выражение конструирования массива;
- выражение получения класса;
- условное выражение.
- Выражение цикла

Данный список неполон и может варьироваться в зависимости от конкретного языка, однако дает общее понятие о базовом выражении языка. Введём несколько определений:

Опр. 1: Выражением будем называть одно из базовых выражений, а также произвольную суперпозицию описанных выше выражений.

Опр. 2: Будем говорить, что выражение S есть подвыражение выражения E, если выражение E есть суперпозиция выражения S и некоторых других выражений.

Опр. 3: Выражение S есть собственное подвыражение выражения E, если S есть подвыражение выражения E и не равно E.

2. Предлагаемые понятия

В работе [4] предлагаются два понятия, характеризующие пригодность метода модульному тестированию и сложность введения метода в средства тестирования:

- внутренний экземпляр класса;
- исходящая зависимость.

Далее детально рассматриваются возможные применения этих идей и дальнейшее их развитие.

3. Внутренний экземпляр класса

3.1. Определения

Определим понятие внутреннего экземпляра.

Опр. 4: Внутренний экземпляр класса – экземпляр класса, созданный оператором конструирования внутри тестируемого метода, либо являющийся результатом вызова метода внутреннего экземпляра.

```
void method() {
    A a = new A();
}
```

Здесь локальная переменная a является внутренним экземпляром. Пусть есть выражение вида

```
b = a.<methodCall>*.
```

Переменная b также будет содержать внутренний экземпляр, если a есть внутренний экземпляр. Не существует возможности изменить характер внешности экземпляра b, если a внутренний экземпляр. Однако если a внешний экземпляр, то он полностью контролируется. Заменяя экземпляр a тестовым двойником [5], можно добиться, чтобы b был произвольным.

Опр. 5: Говорят, что характер внешности b зависит от характера внешности a, если имеет место присваивание b = a.<methodCall>*.

Такое отношение будем называть отношением зависимости характера внешности. Если в качестве вершин взять переменные, которые встречаются в методе, а в качестве отношения смежности взять отношение зависимости характера внешности, то можно построить граф. Рассмотрим пример исходного кода и граф над отношением зависимости характера

внешности для этого участка кода.

```
void method() {
    A a = new A();
    b = a.getValue().getSomeData();
    c = a.getData();
}
```

Для этого метода граф зависимостей характера внешности представлен на рисунке 6.

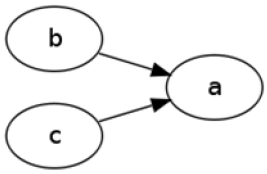


Рис. 6 – Граф зависимостей характера внешности

3.2. Временная составляющая

В реальном исходном коде переменной могут присваиваться разные значения в разные моменты времени. Это осложняет анализ структуры графов зависимостей характера внешности. Из одной переменной может идти несколько дуг. По одному пути переменная может быть внутренним экземпляром, по другому – внешним. Рассмотрим следующий участок кода и иллюстрацию.

```
void method(X x) {
    A a = new A();
    b = a.getValue().getSomeData();
    a = x.getData();
}
```

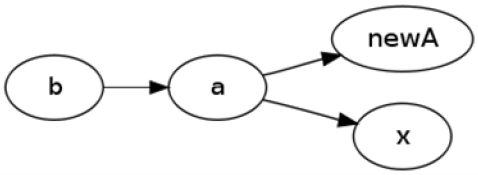


Рис. 2 – Случай повторного присваивания

Из рисунка 2 видно, что граф содержит неоднозначность. В разные моменты времени a имеет разные характеры внешности. Для решения данной проблемы предлагается приводить метод к SSA форме или форме единственного присваивания.

Таким образом если привести тестируемый метод к SSA форме[6], то можно построить граф зависимостей характера внешности и анализ временной составляющей не будет представлять труда. Определим граф внутренних экземпляров следующим образом.

Опр. 6: Граф внутренних экземпляров – граф зависимостей характера внешности, построенный над SSA формой метода.

Понятие внутреннего экземпляра позволяет сформулировать достаточное условие тестируемости метода.

Опр. 7: Для того чтобы метод был тестируем достаточно, чтобы число внутренних экземпляров было равно нулю.

И действительно, если метод имеет нулевое количество внутренних экземпляров, то все экземпляры классов, с которыми работает метод являются внешними и таким образом легко симулировать внешнее окружение, используя тестовые двойники (Test Double) [5]. Метод, удовлетворяющий достаточному условию тестируемости, будем называть абсолютно тестируемым методом.

Опр. 8: Метод абсолютно тестируем, если число внутренних экземпляров равно нулю.

Понятие внутреннего экземпляра дает иное, более формальное определение объектного шва. Такое определение позволяет использовать его для автоматического поиска швов и дальнейшего наглядного представления.

3.3. Категория заменяемости

Не всякий внешний экземпляр может быть заменяем. Поэтому также важна категория заменяемости. Важно рассматривать граф зависимостей характера внешности в пределах всего класса. Экземпляры могут зависеть от экземпляров следующего типа:

- локальные переменные;
- аргументы метода;
- поля класса;
- статические поля других классов;
- статические методы других классов (по сути глобальные поля и методы).

Наиболее легко заменяемы аргументы метода. Заменяемость полей класса разнится в зависимости от их модификатора видимости. Поля с модификатором видимости public могут быть выставлены прямо перед вызовом тестируемого метода. Чуть более сложны в замене protected поля, но для их прямой установки можно воспользоваться паттерном модульного тестирования Test Specific Subclass[5]. Наиболее сложны в замене private поля. Для анализа их характера внешности требуется поиск пути от данного поля до аргумента метода или конструктора в графе внутренних экземпляров.

Граф внутренних экземпляров некоторого класса может выглядеть следующим образом.

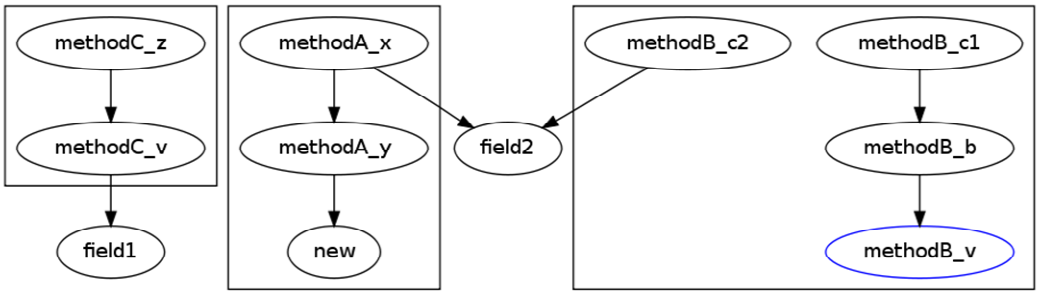


Рис. 3 – Граф внутренних экземпляров класса

На рисунке 3, овалом methodB_v обозначены аргументы метода. Прямоугольники – методы.

Опр. 9: Будем называть экземпляр A гарантированно внешним, если A находится в аргументах метода, либо характер внешности экземпляра A зависит от характера внешности экземпляра B. При этом B есть гарантированно внешний экземпляр

Гарантированно внешние экземпляры попадают в тестируемый класс через аргументы методов и конструкторов. Обращение к статическим полям или методам имеет достаточно ограниченную заменяемость: если для статических полей можно использовать паттерн Supersede Instance Variable[3], то для подмены возвращаемого значения статического метода требуется значительно больше усилий и этот процесс плохо формализуем, в каждом конкретном случае приходится действовать, исходя из конкретных обстоятельств.

3.4. Квантификаторы характера внешности

В связи с наличием в программах операторов ветвления, характер внешности может зависеть от условий, которые встречаются в операторах ветвления или цикла. В связи с этим предлагается ввести квантификаторы характера внешности. Квантификатор – это условие, которое показывает, когда экземпляр будет внешний или внутренний.

4. Исходящая зависимость

4.1. Определения

В отличие от внутренних экземпляров, исходящие зависимости – это классы, экземпляры которых требуются методу для работы, но не обязательно созданы внутри метода. Рассмотрим следующее определение.

Опр. 10: Говорят, что выражение E имеет тип A, если в результате вычисления выражения E результат имеет тип A.

Пусть, например, класс A содержит метод getB, имеющий тип возвращаемого значения B. Тогда выражение a.getB имеет тип B, где a имеет тип A.

Рассмотрим следующий пример.

```
void method(A a) {
 a.getB()
}
```

В данном примере исходящими зависимостями являются классы A и B.

Опр. 11: Класс A есть исходящая зависимость метода M, если существует выражение E, принадлежащее M, такое, что его тип есть A.

Заметим, что возвращаемое значение метода getB() может быть null, и это никак не повлияет на работоспособность метода. Метод не обращается к внутренней структуре возвращаемого значения, тогда как a null быть не может, что требует от создателя теста на метод method подачи ненулевого значения переменной a. Такие зависимости назовем существенными.

Опр. 12: Класс A есть существенная исходящая зависимость метода M, если A – исходящая зависимость метода M и есть обращение к полям или методам A.

В отличие от внутренних экземпляров, количество исходящих зависимостей не может быть нулем. Так или иначе, метод должен взаимодействовать с окружающей средой. Однако предполагается, что число различных исходящих зависимостей не должно превышать некоторого порогового значения. Исходящие зависимости метода образуют множество исходящих зависимостей метода. У правильно спроектированного метода, соблюдающего принцип единственной ответственности, множество исходящих зависимостей логически связанно или, другими словами, обладает высоким зацеплением [7].

Опр. 13: Множество исходящих зависимостей метода M – множество классов A таких, что A есть исходящая зависимость метода M.

4.2. Граф исходящих зависимостей

Для наглядного представления исходящих зависимостей внутри проекта предлагается строить граф исходящих зависимостей. В отличие от графа связности (coupling), граф исходящих зависимостей отражает не наличие знания одного класса о другом, а требование существования экземпляра. Дуга между классом A и классом B существует тогда, когда существует такой метод M класса A, что исходящая зависимость метода M есть класс B.

Выводы

Введенные понятия оказывают существенную помощь при оценке пригодности кода к модульному тестированию. Формальность описанных критериев позволяет реализовать автоматический подсчет внутренних экземпляров и исходящих зависимостей. Наряду с метриками, предложенными в [8,9], они позволяют опосредованно оценивать состояние проекта. На основе описанных понятий разрабатывается приложение для автоматизации подсчета внутренних экземпляров, исходящих зависимостей и построения соответствующих графов. Приложение работает с кодом языка Java и реализовано в виде цели системы сборки Ant.

Литература

1. Бек К. Экстремальное программирование. СПб: Питер, 2002. 224 р.
2. Мартин Фаулер. Рефакторинг. Улучшение существующего кода. СПб: Символ-плюс, 2009.
3. Майкл Физерс. Эффективная работа с унаследованным кодом. М: Вильямс, 2009.
4. Виктор Денисов. Критерий тестируемости исходного кода. // RSDN Magazine. 2010. № 2.
5. Джерард Месарош. Шаблоны тестирования xUnit. СПб: Символ-плюс, 2009.
6. Bilardi G., Pingali K. The Static Single Assignment Form and its Computation. 1999.
7. Stevens, W.P. Structured design // IBM Systems Journal. 1974. Vol. 13, № 2. P. 115-139.
8. McCabe T.J. A Complexity Measure // Software Engineering, IEEE Transactions on. Vol. 2, № 4. P. 308-320.
9. Chidamber S.R., Kemerer C.F. A metrics suite for object oriented design // IIEEE Trans. Software Eng. 1994. Vol. 20, № 6. P. 476-493.

0004.415.25

СТРУКТУРА ИНФОРМАЦИОННО-АНАЛИТИЧЕСКОЙ СИСТЕМЫ ВУЗОВСКОГО ЦЕНТРА ДОПОЛНИТЕЛЬНОГО ОБРАЗОВАНИЯ

Штырова Ирина Анатольевна, ассистент, Балаковский институт техники, технологии и управления (филиал) Саратовского государственного технического университета, Россия, Балаково, comtech@bittu.org.ru

Виштак Ольга Васильевна, д.п.н., профессор, Балаковский институт техники, технологии и управления (филиал) Саратовского государственного технического университета, Россия, Балаково, comtech@bittu.org.ru

В Концепции Федеральной целевой программы развития образования на 2011-2015 годы отмечается необходимость создания современной системы непрерывного образования, подготовки и переподготовки профессиональных кадров. В связи с этим актуальным является развитие системы дополнительного образования в вузе, направленное на решение этой задачи. Дополнительное образование отличается от основного большим объемом динамично изменяющейся информации, так как для него характерны:

- большое количество разнообразных программ повышения квалификации, профессиональной переподготовки, краткосрочных курсов;
- более короткие сроки обучения по сравнению с основными программами профессионального образования;
- индивидуальная траектория обучения для слушателей.

Для обеспечения эффективного управления вузовских центров дополнительного образования (ЦДО) необходимо разработать информационно-аналитическую систему (ИАС), что позволит оптимизировать сбор информации обо всех участниках образовательного процесса, управленческой информации, учебной информации, которая является основой для формирования управленческих решений, анализа педагогической информации, контроля исполнения принятых решений, выдачи справочных, аналитических и отчетных документов, прогнозирования развития центра.

Предваряя процесс определения состава и структуры информационно-аналитической системы ЦДО, следует определить основные типы информации, циркулирующей в педагогической системе ЦДО и являющейся необходимой для эффективной организации управления образовательным процессом ЦДО (ОП ЦДО). В первую очередь надо выделить входную, выходную, внутреннюю, внешнюю информацию.

Входная информация поступает в систему ОП ЦДО или ее подсистемы извне. Это нормативные документы, образовательные программы, научно-педагогическая и учебная информация. Выходная информация формируется в самой системе ОП ЦДО и включает сведения: обо всех участниках образовательного процесса, включая информацию о состоянии и результатах обучения; об учебно-методическом обеспечении; о материально-технической базе. Одна и та же информация может являться входной для одной подсистемы, а для другой, ее вырабатывающей, – выходной. По отношению к субъектам системы ОП ЦДО (преподавателям и слушателям) информация может быть определена и как внешняя, и как внутренняя. Так, например, информация о текущих результатах обучения слушателей является внутренней, а в то же время для сотрудников учебно-методического отдела вуза – внешней.

По содержательному компоненту в системе ОП ЦДО выделяем:

- директивную и нормативную информацию (нормативная документация по организации дополнительного образования, положение о ЦДО, учебные планы образовательных программ и т.д.);
- педагогическую информацию (информацию о состоянии и результатах обучения, о состоянии научной работы преподавателей, о воспитании и развитие обучающихся по образовательным программам подготовки к поступлению в вуз),

- методическую информацию (методические рекомендации по ведению учебного процесса, методические пособия, рекомендации по всем видам контрольных мероприятий);
- учебную информацию (учебники, учебные пособия, справочники и т. д.);
- научную информацию (монографии, сборники статей, научные периодические журналы, тезисы конференций и т. д.);
- сведения о материально-технической базе.

В функциональном отношении содержание информации определяется ее значением для различных этапов управления. В первую очередь выделяем информацию, необходимую для формирования целей и задач, которые затем трансформируются в задачи субъекта обучения, что стимулирует их активность, вызывает позитивные изменения в их личностном развитии.

Далее выделяем плановую информацию, содержащую сведения о параметрах субъектов управления на будущий период; нормативно-справочную информацию, регламентирующую деятельность преподавателя по организации учебного процесса.

Для педагогического анализа, контроля, прогнозирования необходима учетная (тематическая) информация, которая характеризует деятельность субъектов ОП ЦДО за определенный прошедший период времени. На основании этой информации проводится коррекция плановой информации, анализ деятельности преподавателей и слушателей, принимаются решения по более эффективному управлению ОП ЦДО.

В оперативном управлении используется текущая (оперативная) информация, характеризующая ОП ЦДО за текущий период времени. К оперативной информации предъявляются повышенные требования по скорости поступления и обработки, а также по степени ее достоверности, в связи с тем, что временной диапазон обучения слушателя по программам дополнительного образования достаточно ограничен. И от того, насколько быстро и качественно проводится ее обработка, во многом зависит эффективность ОП ЦДО. Тематическая и текущая информация по своему содержанию является компонентом итоговой информации.

Отношения между субъектами образовательного процесса предполагают наличие информационных взаимосвязей между ними. Информационные связи между получателями и отправителями информации разнообразны и сложны. В зависимости от уровня, на котором находятся субъекты коммуникационного процесса, выделяют горизонтальные и вертикальные связи на основе прямого и косвенного общения.

Первичной формой общения в учебном процессе является непосредственное обучение слушателей, которое представлено в вертикальном и горизонтальном общением. Устная коммуникация является наиболее эффективной с точки зрения преподавателя по отношению к слушателю, так как позволяет оперативно донести информацию и практически сразу установить обратную связь. Однако, учитывая специфику образовательной программы, например, рассмотрим обучение компьютерным технологиям, необходимо включать в учебный процесс различные виды самостоятельной работы. Так, например, при обучении по образовательным программам подготовки к поступлению в вуз, повышению квалификации и профессиональной переподготовки предполагается выполнение проектных работ. Поэтому становится предпочтительной письменная коммуникация, так как она обеспечивает более высокую степень чистоты передачи информации, позволяет сохранять ее в неизменном виде в течение неопределенного времени. При этом информационное взаимодействие между субъектами ОП ЦДО осуществляется в опосредованной форме.

Таким образом, анализ информации, циркулирующей в ЦДО, показывает сложность и многообразие информационных потоков, необходимость интегрирования сведений о субъектах ОП ЦДО в единую информационную систему, объединяющую информационно-справочные сведения, информацию об управлении учебным процессом, результаты мониторинга и анализа образовательного процесса.

В связи с этим нами предлагается структура ИАС ЦДО, которая включает в себя четыре подсистемы: информационно-справочную, управления учебным процессом, мониторинга, аналитическую (рис.1).

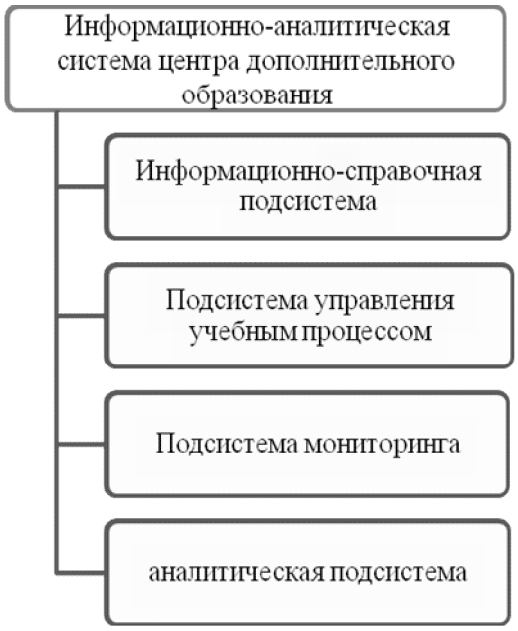


Рис. 1 – Структура информационно-аналитической системы вузовского центра дополнительного образования

Информационно-справочная подсистема предназначена для хранения информации:

- по слушателям:
  - анкетные данные слушателя;
  - данные об образовании слушателя;
  - результаты обучения;
- по преподавателям:
  - анкетные данные;
  - данные о квалификации;
  - рейтинг преподавателя;
- по процессу обучения.

Пользователями информации, хранящейся в информационно-справочной подсистеме, являются все участники образовательного процесса, но с установлением ограничений по доступу.

Подсистема управления учебным процессом предназначена для формирования управленческих решений, которые документировано представлены:

- в учебных планах образовательных программ;
- в планах распределения нагрузки;
- в индивидуальных учебных программах слушателей;
- расписании учебных занятий;
- в ведомостях занятий групп и индивидуальных слушателей;
- в индивидуальных картах слушателей и протоколах аттестационных комиссий (результаты обучения).

Пользователями информации, хранящейся в подсистеме управления учебным процессом, являются все участники образовательного процесса, но с установлением ограничений по доступу.

Подсистема мониторинга предназначена для систематического контроля всей деятельности ЦДО, образовательного процесса, выполнения планов, программ и

формирования оперативной информации об их состоянии. Пользователями информации, хранящейся в подсистеме мониторинга, являются сотрудники ЦДО.

Аналитическая подсистема предназначена формирования отчетных документов аналитических справок, прогнозов развития деятельности ЦДО. Пользователями информации, хранящейся в аналитической подсистеме, являются все участники образовательного процесса, но с установлением ограничений по доступу.

Таким образом, внедрение в образовательный процесс вузовского центра дополнительного образования информационно-аналитической системы позволит решить двуединую задачу: оптимальное обеспечение информационной поддержки управленческой и педагогической деятельности преподавателей и обеспечение информационной поддержки учебной и управленческой деятельности слушателей.

УДК 628.339.066.1

ИСПОЛЬЗОВАНИЕ СУБД ДЛЯ ВЫБОРА ИНДИВИДУАЛЬНЫХ ФИЛЬТРОВ ВОДООЧИСТКИ

Кравчик Владислав Георгиевич, к.т.н. доцент, Южно-Российский государственный университет экономики и сервиса, Россия, Шахты
Кокарев Иван Викторович, ст. пр., Южно-Российский государственный университет экономики и сервиса, Россия, Шахты
Тряпичкин Сергей Александрович, ст.пр., Южно-Российский государственный университет экономики и сервиса, Россия, Шахты, telescii@mail.ru

Большинство населенных пунктов (города, поселки и т.п.) не может позволить большие затраты на индивидуальные системы водоочистки в жилом секторе. Населению приходится самостоятельно решать вопросы дооборудования внутриквартирных водопроводных сетей устройствами дополнительной очистки воды. Далеко не всегда вода в точке водопотребления, доставляемая к индивидуальному потребителю, из-за изношенности водопроводных сетей удовлетворяет. Поэтому качественное водоснабжение является задачей, решаемой самим индивидуальным потребителем.

Одним из способов решения этой проблемы является установка индивидуальных фильтров для очистки воды. Доочистка воды непосредственно у индивидуального потребителя позволяет решить целый ряд задач по улучшению качества жизни.

На рынке представлено большое разнообразных систем и устройств очистки воды. Существующие системы высококачественной водоподготовки стоят дорого и не всегда могут быть применены в малогабаритном жилье. Недорогие малогабаритные фильтры, которые возможно установить, в этом случае не обеспечивают хороших технико-экономических показателей. Актуальной является задача выбора такого фильтра, который, с одной стороны, малогабаритный, с другой стороны, обеспечивает регламентированные критерии качества.

Необходимо иметь объективную систему, которая позволяет в условиях большого ассортимента выпускаемых моделей, выбрать малогабаритный фильтр применительно к конкретным условиям точки водопотребления. Такой объективной системой, безусловно, является СУБД, которая включает характеристики как места водопотребления, так и характеристики воды на входе конкретного водопотребителя и характеристики существующих средств водоочистки.

СУБД позволяет создать автоматизированную систему, учитывающую множество параметров и характеристик фильтров, параметров системы водоснабжения, параметров водопотребления, параметров качества воды. Задавать различные критерии выбора, фильтровальной системы в целом и каждого из её компонентов в отдельности. Более того, в этом случае четко обозначаются наиболее узкие с точки зрения конструктивных особенностей недостатки имеющегося в доступе оборудования, что позволяет наметить

наиболее перспективные пути совершенствования этого оборудования за счет включения в алгоритмы его функционирования новых физических эффектов, материалов и режимов работы.

Идея была реализована при помощи программы Access. Была создана база данных, которая позволила сформировать выборку или единичное устройство по разнообразным критериям. В качестве критериев были использованы:

- критерий «Качество воды в регионе»;
- критерий «Класс фильтра»;
- критерий «Принцип работы»;
- критерий «Конструкция фильтра»;
- критерий «Типоразмер фильтра»;
- критерий «Фильтрующий материал».

На данном этапе при формировании обобщенного критерия оптимизации выбора сформированы составляющие его частные критерии, которые приведены выше в порядке их приоритетности (рис.1.).

На основании этих критериев СУБД позволяет выбрать по заданному значению обобщенного критерия индивидуальный фильтр, максимально учитывающий особенности конкретного водопотребителя (квартира, коттедж, офис).

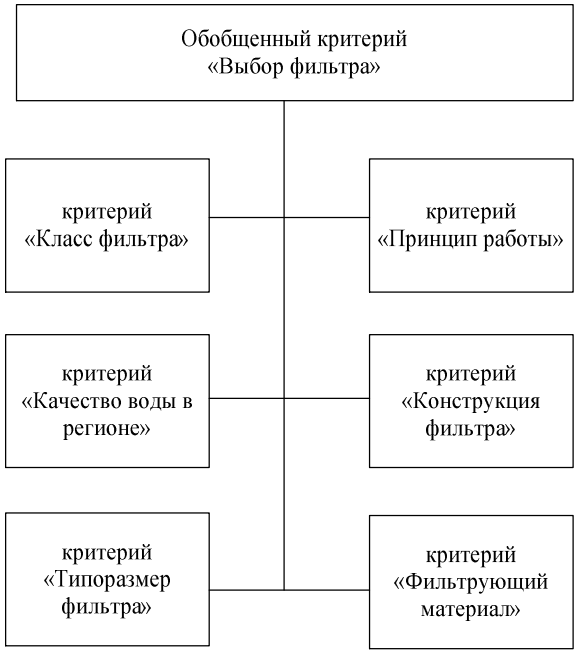


Рис. 1 – Структура критериев СУБД для выбора индивидуальных фильтров водоочистки

Уже на первом этапе применения СУБД выяснилось, что в малогабаритных недорогих фильтрах отсутствует система возобновления фильтрующих свойств и системы утилизации и обеззараживания фильтросадка.

Анализ современных достижений науки в области очистки воды показывает, чт одним из наиболее перспективных и прогрессивных путей развития способов очистки воды для бытового потребителя является способ, включающий в себя применение ультразвуковых колебаний высокой интенсивности. Обусловлено это тем, что ультразвуковые колебания оказывают влияние практически на все известные микроорганизмы, позволяют эффективно очищать и обеззараживать поверхности фильтра и фильтрэлемента. Особенностью способа является то, что ультразвуковое излучение проникает по всему объему развитой поверхности фильтрэлемента, что позволяет эффективно очищать и обеззараживать как фильтросадок и фильтрэлемент, так и внутреннюю поверхность фильтра.

Ультразвуковое воздействие характеризуется степенью развитости кавитации. Основные закономерности кавитационного разрушения поверхностных пленок установлены

достаточно определенно. Известно, что удаление пленок происходит не только вследствие эрозии под действием микроударных нагрузок, возникающих при захлопывании кавитационных пузырьков, но и в результате действия пульсирующих пузырьков, стабильно существующих в звуковом поле в течение длительного времени. Известно, что существует взаимосвязь между интенсивностью кавитационного разрушения твердых тел в звуковом поле и физическими свойствами жидкости, а также параметрами звукового поля [1]. Также известны экспериментальные исследования, показывающие, что при воздействии ультразвуковых колебаний уменьшается толщина пограничного слоя вследствие образования в нем вихревых микропотоков, а это приводит к ускорению протекания процессов диффузии и массообмена в пограничном слое и, следовательно, облегчает растворение пленки загрязнений, а также улучшает условия химического взаимодействия загрязнения жидкостью.

Загрязнения, которые возможно удалить при помощи ультразвуковой обработки, классифицируются по:

- способности противостоять воздействию микроударных нагрузок;
- прочности связи с очищаемой поверхностью;
- характеру химического взаимодействия с жидкостью.

По первому признаку загрязнения подразделяются на кавитационно–стойкие и кавитационно–нестойкие, по второму – на прочно- и слабосвязанные с очищаемой поверхностью, по третьему – на химически взаимодействующие и невзаимодействующие с моющей жидкостью [2]. Жировые пленки кавитационно–стойкие, слабо связаны с очищаемой поверхностью, химически взаимодействуют с моющей жидкостью. Продукты коррозии кавитационно–нестойкие, прочно связаны с очищаемой поверхностью. Смолистые осадки кавитационно–стойкие, прочно связаны с очищаемой поверхностью.

Поверхностные пленки в жидкости под действием ультразвука разрушаются вследствие кавитации и акустических течений. Наряду с кавитацией и акустическими течениями известную роль в ускорении движения частиц может играть радиационное давление. Определенную роль могут играть знакопеременные напряжения, возникающие в пленке загрязнений при изгибных колебаниях озвучиваемой части фильтра, способствующие отслаиванию и разрушению пленки, если ее усталостная прочность незначительна.

Известно, что интенсивность кавитации, скорость и характер акустических течений, величина радиационного давления, амплитуды колебаний самой детали зависят от частоты и интенсивности звукового поля, физических свойств жидкости, величины внешнего статического давления.

Пример характеристик фильтрующих материалов, вводимых в СУБД, приведен в таблице 1. Параметры фильтра

Таблица 1. Зернистые, природные фильтрующие материалы

Вид материала	Плотность, г/см^3	Пористость загрузки, %
Кварцевый песок	2,6 – 2,65	40 – 42
Антрацит дробленый	1,6 – 1,7	45
Керамзит дробленый	1,2 – 1,5	58 – 62
Керамзит недробленый	1,7 – 1,8	45
Шунгизит дробленый	1,5 – 1,8	56 – 58
Доменные шлаки	2,6	42 – 44
Цеолит	2,18 – 2,5	25 – 28
Активированный уголь	0,3 – 0,42	65

Аналогичные таблицы составляются для других частных критериев (жилищные условия, качество воды, стоимость, габариты устройства и т.д.), как, например, в таблице 2, где представлены физико-механические свойства природных, искусственных и синтетических волокон.

Фрагмент таблицы базы данных приведен на рис.2. Подобным образом сформированы

таблицы по другим частным критериям, отличающиеся только количеством вариантов и числом классификационных признаков.

Таблица 2. Физико-механические свойства природных, искусственных и синтетических волокон

Волокна	Разрывная длина (в км)		Разрывная прочность (в кг/мм^2)
	В сухом состоянии	В мокром состоянии	
Природные волокна			
Хлопок	27-36	30-40	41-54
Шелк	27-32	22-28	35-42,5
Шерсть	12-14	10-12	14-18,5
Искусственные волокна			
Вискозное	15-20	7-9	22,5-30
Высокопрочное вискозное	40-50	30-40	60-75
Ацетатное	10,5-14	6-6,5	15-18,5
Синтетические волокна			
Перхлорвиниловое (хлорин)	15-20	15-20	24-32
Полиамидное (капрон, анид, нейлон)	45-70	40-67	51-59
Полиэфирное (лавсан)	35-55	35-55	49-77
Полиакрилонитрильное (нитрон)	25-40	25-40	28-47
Полипропиленовое	30-55	30-55	36-48

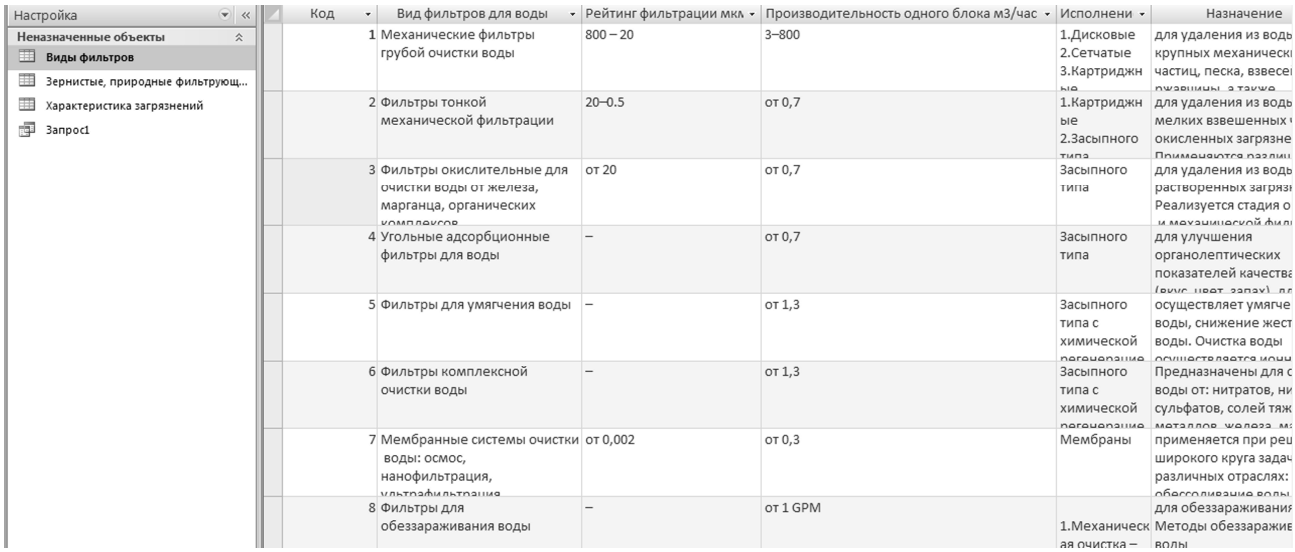


Рис. 2 – Фрагмент таблицы «Виды фильтров»

Таким образом, применение данной СУБД в системах обеспечения индивидуальных потребителей позволяет на основании минимизации соотношения обобщенного критерия цена/качество или обобщенного критерия цена/габаритные размеры выбрать наилучшее решение для индивидуального потребителя. СУБД применялась для выбора конкретной модели фильтра на уровне возможностей внутригородского рынка, внутри областного рынка и интернет рынка Рунета.

Литература

1. Наука. Гл. ред. физ.-мат.лит., 1988. – (Соврем. пробл. Физики). – 160 с. Под. ред. Розенберга Л.Д., Физика и техника мощного ультразвука. Физические основы ультразвуковой технологии. М.: Наука. 1970. – 685 с.
2. Самуйлов, В. С. КОРЕЛЛЯЦИОННЫЕ ЗАВИСИМОСТИ ДЛЯ СКОРОСТИ ЗВУКА В БИНАРНЫХ СМЕСЯХ *Н*-АЛКАНОВ.// В.С. Самуйлов, А. П. Щемелёв, О. Г. Поддубский, М. А. Екимова. Ультразвук и термодинамические свойства вещества.-2009.-№36. С. 67-17.

УДК 004.45

РАЗРАБОТКА СИСТЕМЫ АВТОМАТИЗАЦИИ РЕГРЕССИОННОГО ТЕСТИРОВАНИЯ

Иконников Виктор Викторович, аспирант, Южно-Уральский Государственный Университет, Россия, Челябинск, viktor.ikonnikov@gmail.com
Лебедев Александр Анатольевич, аспирант, Южно-Уральский Государственный Университет, Россия, Челябинск, a.a.lebedev@gmail.com

Введение

Одной из наиболее важных проблем российского рынка программного обеспечения является низкая конкурентоспособность российской информационно-технологической продукции на международных рынках, а также низкий уровень развития отечественных инструментальных средств и методологий разработки программного обеспечения.

Главная цель развития отрасли информационных технологий России – преобразование ее в динамично развивающуюся, высокотехнологичную, эффективную и конкурентоспособную отрасль, интегрированную в мировую экономику в рамках международного разделения труда.

Для достижения указанной цели необходимо решить следующие задачи:

- повышение качества разрабатываемого программного обеспечения;
- совершенствование технологий разработки программного обеспечения.

Промышленные предприятия России вступают в процесс активной модернизации производства. Замена оборудования и систем автоматического управления технологическими процессами, внедрение систем управления эффективностью производства, а также интегрированных систем управления предприятием влечет за собой обновление информационно-вычислительных систем. Мировая практика показывает, что наиболее надежными вычислительными системами являются мейнфреймы.

Мэйнфрейм – это вычислительная системы, ориентированная на бесперебойное исполнение исключительно больших, смешанных рабочих нагрузок при высоком уровне коэффициента использования системы, обеспечивающие защиту информации, максимальную производительность и высокую пропускную способность.

В настоящий момент на большинстве крупных промышленных предприятиях Европы, США и Японии используются мейнфреймы от компании IBM (80% рынка мейнфреймов), работающие под управлением операционной системы z/OS.

Развитие рынка мейнфреймов в России отстает от среднемировых показателей, однако, в последние 2 года идет лавинообразный всплеск интереса к разработке программного обеспечения для мейнфреймов.

Промышленные предприятия предъявляют высокие требования не только к самим вычислительным системам, но и к программному обеспечению, которое в первую очередь должно отвечать критерию надежности и стабильной работы в условиях непрерывного технологического процесса [1]. Достижение заданных показателей качества возможно лишь при правильном построении процесса жизненного цикла разработки ПО, а особенно – процесса тестирования.

Руководители и разработчики начинают понимать важность процесса тестирования, для повышения качества программных систем. Становится очевидным, что чем позже начать тестировать программную систему, тем выше риски и тем менее надежной она может получиться. Тестирование из прикладного процесса с невысоким приоритетом переходит в разряд особо важных процессов, чей жизненный цикл начинается параллельно с разработкой программных систем [2].

Наиболее эффективным способом поиска ошибок считается регрессионное тестирование программного обеспечения, а наиболее универсальной стратегией

тестирования ПО является тестирование «черного ящика», т.е. тестирование в соответствии с заранее заданными требованиями без обращения к исходному коду программы [3].

На данный момент существует множество систем автоматизации тестирования, однако, большинство из них ориентировано тестирование программного обеспечения, разрабатываемого для PC-совместимых компьютеров. Сложность программного обеспечения для мейнфреймов, его глубокая интеграция с операционной системой z/OS, ориентированность на консольный интерфейс, а также отсутствие на рынке программного обеспечения готовых решений автоматизации регрессионного тестирования ПО с закрытым исходным кодом вынуждают производителей программного обеспечения разрабатывать собственные средства автоматизации тестирования [4].

Отсутствие универсальных систем автоматизации регрессионного тестирования заставляет разработчиков вкладывать значительные средства в разработку собственных систем, ориентированных на конкретный продукт.

Стремительный рост числа компаний, занимающихся разработкой программного обеспечения для мейнфреймов, работающих под управлением операционной системы z/OS, а также осознание эффективности регрессионного тестирования как неотъемлемого этапа жизненного цикла программного обеспечения, являются очевидными предпосылками разработки комплексной методологии, а также универсального инструмента автоматизации регрессионного тестирования программного обеспечения с закрытым исходным кодом.

1. Постановка задачи

Система должна позволить автоматизировать процесс тестирования программного обеспечения, разрабатываемого для мейнфреймов, работающих под управлением операционной системы z/OS.

В веб-интерфейсе системы автоматизации тестирования необходимо реализовать набор инструментов, позволяющих упростить процессы управления тестированием и ручной обработки результатов тестирования.

Для расширения стандартного функционала системы предусмотрен прикладной интерфейс программирования API, позволяющий переопределять стандартные модули системы, а также реакцию системы на определенный класс событий.

Для ускорения процесса тестирования система должна обладать возможностью перераспределения нагрузки между мейнфреймами с тестируемым программным обеспечением.

2. Структура системы

Система автоматизации спроектирована как распределенное клиент-серверное приложение и конструктивно может быть разделена на три части: сервер, клиент и веб-интерфейс. На приведенном ниже рисунке 1 представлена упрощенная структура связей между составляющими её компонентами.

Клиент используется для запуска регрессионного тестирования и представляет собой копию тестирования, запущенного в данный момент на одном из мейнфреймов. Выполняемые клиентом функции:

1. Чтение и проверка конфигурационных файлов.
2. Проверка целостности структуры проекта.
3. Проверка и запуск скриптов в “умных”-пакетах тестов.
4. Генерация тестов.
5. Осуществление подстановок в шаблонах тестов.
6. Объединение и слияние тестов для различных версий тестирумеого программного продукта.
7. Запуск регрессионного тестирования.
8. Сохранение результатов тестирования.
9. Обработка результатов тестирования.
10. Оповещение тестеров о возникающих ошибках и вариантах их решения.

11. Запрос разрешения сервера на выполнение операций запуска тестов на удаленном мейнфрейме.
12. Регистрация событий на сервере.

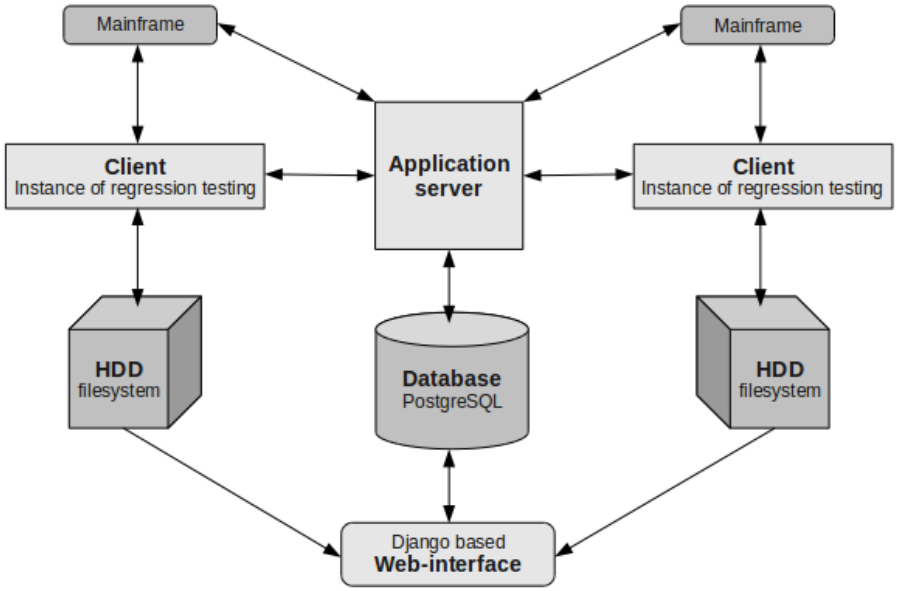


Рис. 1 – Структура системы автоматизации тестирования

Основное предназначение сервера – контроль работы клиентов. Выполняемые сервером функции:

1. Регистрация действий осуществляемых клиентом.
2. Выдача разрешения клиенту на выполнения тех или иных действий.
3. Осуществление контроля загруженности мейнфреймов.
4. Отмена запущенных задач при возникновении конфликтов.
5. Оповещение администратора при возникновении сбоев.

Контроль нагрузки осуществляется путем опроса программы мониторинга системных ресурсов мейнфрейма MXI и анализа полученных данных.

В основе архитектуры клиентской части системы автоматизации регрессионного тестирования будет заложен один из современных принципов построения программного обеспечения – принцип Модель-Вид-Контроллер (MVC), что позволяет разделить модель данных приложения, пользовательский интерфейс и управляющую логику на три отдельных компонента так, что модификация одного из компонентов будет оказывать минимальное воздействие на другие компоненты.

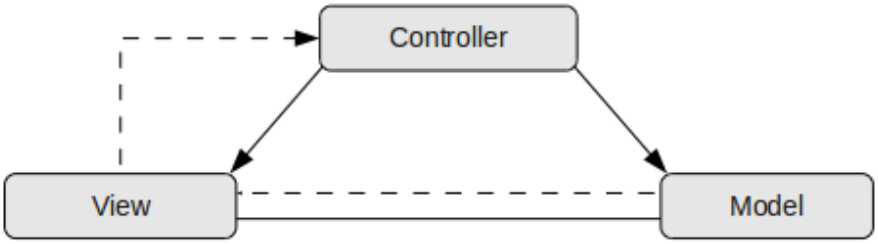


Рис. 2 – Архитектура Модель-Вид-Контроллер

В архитектуре разрабатываемой системы представление отвечает за генерацию исходных тестов и отчетов о тестировании по их абстрактному описанию и шаблонам. Основное предназначение модели – сохранение и анализ результатов тестирования, а также реакция на внутрисистемные события. Контроллер осуществляет настройку окружения как для отдельного тесткейза так и для всего тестирования в целом, поддержку целостности и работоспособности окружения, управление ходом выполнения тестирования, получение и интерпретацию результатов тестирования, а также информирование модели о необходимости реакции на внутрисистемные события.

Использование архитектуры MVC, а также выделение интерфейса прикладного программирования API, позволяющего конечному пользователю переопределить стандартные функции и реакцию системы на распространяемые в ней события, позволяет расширить базовый функционал системы, делает её гибкой и универсальной системой, способной решать широкий класс задач в области автоматизации тестирования.

Веб-интерфейс системы автоматизации тестирования предназначен для управления процессом тестирования, а также обработки и визуализации результатов тестирования.

3. Реализация системы

Разрабатываемая система автоматизации регрессионного тестирования представляет собой фреймворк, реализуемый на высокоуровневом языке объектно-ориентированного программирования Python. Использование языка Python, в основу которого заложены такие парадигмы программирования, как динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений и удобные высокоуровневые структуры данных, позволяет значительно ускорить процесс разработки и отладки программного обеспечения. Стоит отметить, что язык Python является одним из наиболее популярных языков программирования, используемых техническими специалистами, занимающимися тестированием.

Взаимодействия между клиентами и сервером реализовано на базе технологии удаленного вызова процедур RPC и её реализации на языке Python – библиотеке Pyro (Python Remote Objects).

При решении задачи классификации знаний для комплексного анализа результатов регрессии на основе накопленного опыта используется метод опорных векторов для случая линейной классификации при заданном формальном описании ожидаемого результата. Модули обработки результатов тестирования реализованы на языке C с использованием библиотеки Pyrex.

Для сохранения результатов тестирования и пользовательских настроек используется система управления базами данных PostgreSQL.

Веб-фронтэнд системы реализован на базе веб-фреймворка Django. Для подсветки логов, получаемых с мейнфреймов, а также синтаксиса языка управления запуском пакетных заданий JCL, использована библиотека.

Для взаимодействия системы с тестируемым программным обеспечением реализован программный интерфейс, предоставляющий доступ к подсистеме ввода заданий на мейнфрейм JES2/3 по протоколу FTP.

4. Внедрение системы

Система автоматизации используется QA-инженерами компаний ООО «Прикладные технологии» (РФ) и Rocket Software Incorporated (США) для распределенного тестирования программного продукта Advanced Allocation Optimizer. С момента введения в эксплуатацию было проведено около 350 регрессионных тестирований целевого ПО, в общей сложности запущено около 700000 тестов. На данный момент система работает на кластере из 2 серверов и 7 мейнфреймов. Время наработки на отказ в текущей конфигурации системы составляет 10000 часов.

Литература

1. Автоматизированное тестирование программного обеспечения, Элфрид Дастин, Джефф Рэшка, Джон Пол, Издательство: Лори, 2003 г.
2. Введение в тестирование программного обеспечения, Луиза Тамре, Вильямс, 2003
3. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем, Борис Бейзер, Питер, 2004 г.
4. Автоматизация процессов тестирования, И. Винниченко, Питер, 2005 г.

УДК 004.421

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ОЦЕНКИ КАЧЕСТВА МОЛОЧНОГО СЫРЬЯ[1]

Валитова Елена Геннадьевна, ст. преподаватель, Филиал Московского государственного университета технологий и управления в г. Мелеузе, Россия, Мелеуз, veg02@mail.ru
Шиянова Наталья Ивановна, к.т.н., доцент, Филиал Московского государственного университета технологий и управления в г. Мелеузе, Россия, Мелеуз, shianova58@mail.ru
Мунасыпов Рустем Анварович, д.т.н., профессор, Уфимский государственный авиационный технический университет, Россия, Уфа, rust40@mail.ru

Особую значимость в условиях рыночных отношений приобретает вопрос обеспечения населения высококачественными продуктами питания отечественного производства, в частности, молочными продуктами. Контроль молочного сырья на предприятиях перерабатывающей промышленности является основной задачей, от решения которой полностью зависит производство продукта гарантированного качества. В то же время, технологические процессы в молочной промышленности характеризуются высокой интенсивностью и сложностью. Молочное сырье имеет нестабильные качественные и количественные характеристики, что требует контроля и управления параметрами технологического процесса в режиме реального времени [1].

Оценка сортности молочного сырья на этапе приемки является ключевым звеном в системе качества молочной продукции. Для совершенствования системы оценки сортности молочного сырья необходимы новые подходы, позволяющие повысить достоверность определения сортности молочного сырья. Состав молочного сырья определяется Федеральным Законом № 88-ФЗ от 12 июня 2008 года «Технический регламент на молоко и молочную продукцию», который регулирует процессы производства, реализации молока и продуктов его переработки и вступил в силу с 17 декабря 2008 года. Согласно данному Федеральному Закону, вводится распределение молока на 3 сорта (высший, первый, второй) С момента вступления в действие технического регламента, молоко, называемое по ГОСТ Р 52054-2003 «несортовым», не допускается к переработке.

Молочное сырье сортируют по качеству в зависимости от органолептических, физико–химических и микробиологических показателей. Таким образом, параметрами входного контроля молочного сырья, определяющими его сортность, являются [2]:

- органолептические показатели – вкус, цвет, запах;
- физико-химические показатели – содержание белка, кислотность, жирность, плотность, температура, группа чистоты;
- показатели безопасности – содержание КМАФА и М, патогенных, соматических клеток, токсичных элементов, микотоксинов, антибиотиков, ингибирующих веществ, пестицидов, радионуклидов.

Критерии входного контроля – соответствие свойств сырья установленным для каждого сорта требованиям – позволяют определить сортность молочного сырья. В таблице 1 приведены нормы физико-химических показателей для молока высшего, первого и второго сорта.

Различные аспекты функционирования системы могут быть продемонстрированы с помощью UML–диаграмм. Постановку задачу можно осуществить посредством построения диаграммы вариантов использования для системы оценки качества молочного сырья (рисунок 1). Варианты использования – это функции, выполняемые системой, действующие лица – это заинтересованные лица по отношению к создаваемой системе [3].

При разработке программного продукта использовалась визуальная среда разработки C++ Builder 6.0 и объектно-реляционная система управления базами данных PostgreSQL.

C++ – компилируемый строго типизированный язык программирования общего

[1] Статья рекомендована к опубликованию в журнале "**Автоматизация в промышленности**"

назначения. Поддерживает разные парадигмы программирования: процедурную, обобщённую, функциональную; наибольшее внимание уделено поддержке объектно-ориентированного программирования. C++ Builder – среда быстрой разработки (RAD), выпускаемая компанией Borland. Предназначена для написания программ на языке программирования C++. C++ Builder объединяет Библиотеку визуальных компонентов и среду программирования (IDE), написанную на Delphi с компилятором C++. PostgreSQL – это свободно распространяемая объектно-реляционная система управления базами данных, наиболее развитая из открытых СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных.

Таблица 1. Нормы физико-химических показателей качества молока

Наименование показателя	Ед.измерения	Нормы для молока сорта		
		высшего	первого	второго
Содержание белка	%	≥2,8	≥2,8	≥2,8
Кислотность	°Т	16-18	16-18	19-21
Жирность	%	2,8-6,0	2,8-6,0	2,8-6,0
Плотность	кг /м^3	1028	1027	1027
Температура	°С	6±2	<10	-
Группа чистоты		1	1	2

На рисунке 1 представлена структурная схема организации связи данных.

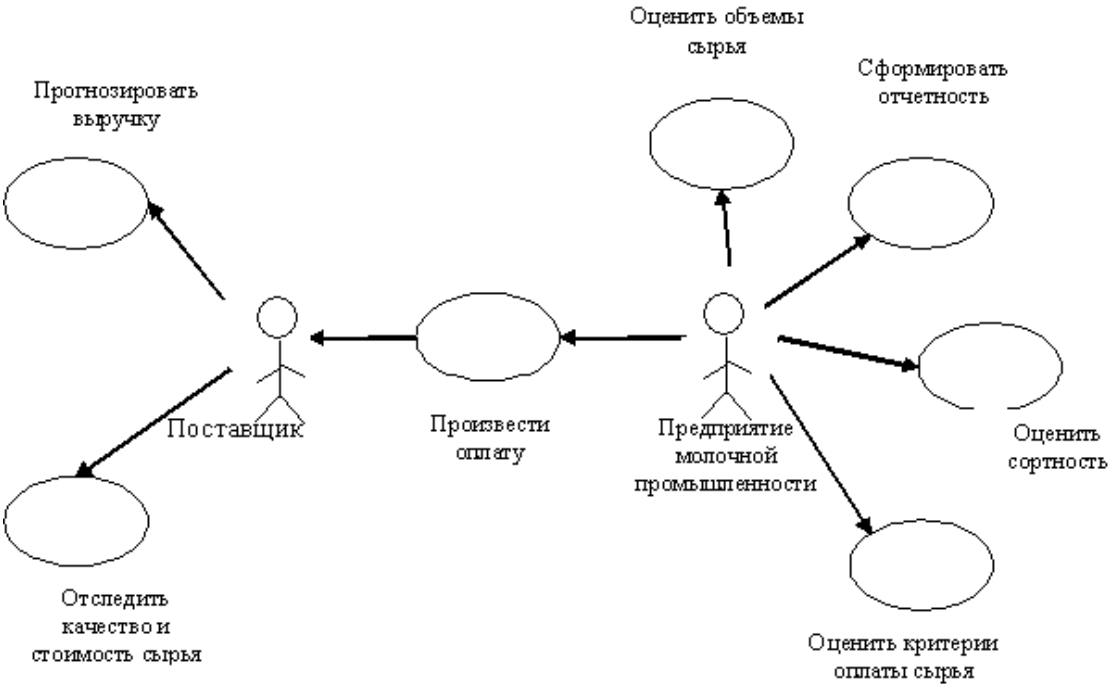


Рис. 1 – Диаграмма вариантов использования

Основные возможности и функциональность:

- Надежность PostgreSQL является проверенным и доказанным фактом и обеспечивается следующими возможностями:
- Наличие механизма протоколирования всех транзакций, что позволяет восстановить систему после возможных сбоев.
- Возможность восстановления базы данных на любой момент в прошлом, что позволяет осуществлять непрерывное резервное копирование кластера PostgreSQL.
- Целостность данных. PostgreSQL поддерживает целостность данных на уровне схемы – это внешние ключи (foreign keys), ограничения (constraints).

- Производительность PostgreSQL основывается на использовании индексов, интеллектуальном планировщике запросов, тонкой системы блокировок, системой управления буферами памяти и кэширования, превосходной масштабируемости при конкурентной работе.
- Расширяемость PostgreSQL означает, что пользователь может настраивать систему путем определения новых функций, агрегатов, типов, языков, индексов и операторов.

Объектная ориентированность PostgreSQL позволяет перенести логику приложения на уровень базы данных, что сильно упрощает разработку клиентов, так как вся бизнес логика находится в базе данных. Функции в PostgreSQL однозначно определяются названием, количеством и типами аргументов.

Доступ программы к базе данных осуществляется с использованием программного интерфейса доступа к базам данных ODBC. Это позволяет единообразно оперировать с разными источниками данных, отвлекаясь от особенностей взаимодействия в каждом конкретном случае.

База данных программы для оценки качества молочного сырья состоит из 11 таблиц, связанных между собой отношениями «один-много» (рисунок 2):

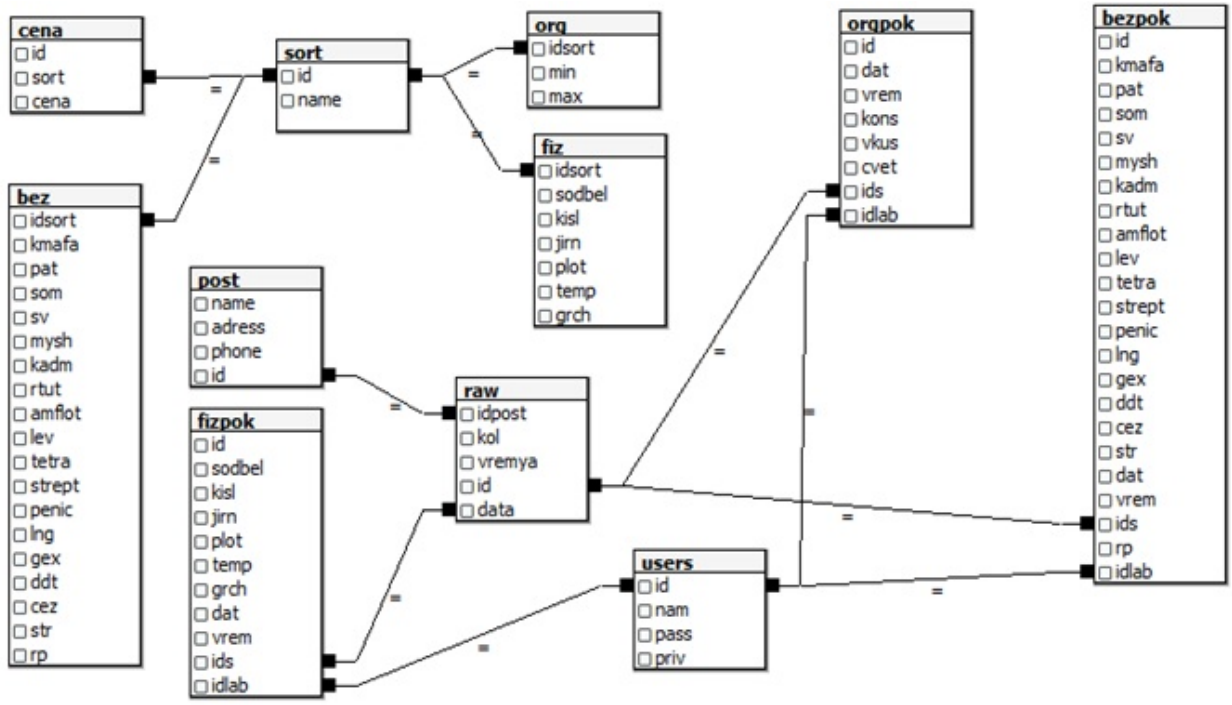


Рис. 2 – Схема организации массивов данных

Структура данных приведена в таблице 2.

Кроме того, используются таблицы: «bez» – эталонные показатели безопасности, «fiz» – эталонные физико-химические показатели сырья, «post» – поставщики сырья, «cena» – стоимость сырья, «sort» – сортность сырья, «org» – эталонные органолептические показатели, «bezpok» – эталонные показатели безопасности молочного сырья, «raw» – сводные данные о партии сырья.

Определение сортности молочного сырья осуществляется путем сопоставления физико–химических, органолептических показателей и показателей безопасности с нормами для молока высшего, первого и второго сортов.

Программная реализация оценки качества молочного сырья позволяет обеспечивать оперативный контроль за качеством поставляемого сырья как со стороны персонала перерабатывающего предприятия пищевой промышленности, так и со стороны поставщиков сырья. В результате достигается повышение количества высококачественного сырья, расширение ассортимента выпускаемой продукции с высокими потребительскими свойствами.

Таблица 2. Фрагменты метаописания таблиц

Название таблицы	Имя поля	Тип данных	Назначение
«users» – данные о пользователях системы	Id	Счетчик	Содержит уникальный идентификационный номер пользователя, что позволяет связать данную таблицу с полем «idlab» из таблиц «fizpok», «orgpok», «bezpok»
	Nam	Текстовый	ФИО сотрудника.
	Pass	Текстовый	Пароль доступа к информации.
	Priv	Числовой	Тип привелегий
«fizpok» – данные об измеренных физико-химических показателях	Id	Счетчик	id номер пробы
	sodbel, kisl, jirn, plot, temp, grch	Вещественный	Данные об измеренных показателях
	Dat	Дата	Дата снятия показаний
	Vrem	Текстовый	Время снятия показаний
	Ids	Числовой	ID поставки
	Idlab	Числовой	ID лаборанта
«orgpok» – измеренные органолептические показатели	Id	Счетчик	Содержит уникальный ИД пробы
	Dat	Дата	Дата снятия показаний
	Vrem	Текстовый	Время снятия показаний
	Kons	Числовой	Консистенция
	Vkus	Числовой	Вкус и запах
	Cvet	Числовой	Цвет
	Ids	Числовой	ID поставки
	Idlab	Числовой	ID лаборанта

Литература

1. Ивашкин Ю.А., Протопопов И.И., Бородин А.В., Копчиков А.В., Шутов С.А. Моделирование производственных процессов мясной и молочной промышленности / Ю.А. Ивашкин, И.И. Протопопов, А.В. Бородин и др.; под редакцией Ю.А. Ивашкина. – М.: ВО «Агропромиздат», 1987. – 232 с.:ил.
2. Краснов И.Н., Краснова А.Ю., Филин В.М., Филин Д.В. Механизация производства, первичной обработки и переработки молока / И.Н. Краснов, А.Ю. Краснова, В.М. Филин, Д.В. Филин. – Ростов н/Д: Терра Принт, 2009. – 388 с.
3. Камаев В.А., Костерин В.В. Технологии программирования: Учебник/ В.А. Камаев, В.В. Костерин. – 2-е изд., перераб и доп – М.:Высш.шк., 2006. – 454 с.:ил..

УДК 519.87

МАТЕМАТИЧЕСКАЯ МОДЕЛЬ ТЕХНОЛОГИЧЕСКИХ ПРОЦЕССОВ НЕПРЕРЫВНЫХ ПРОИЗВОДСТВ (НА ПРИМЕРЕ ТП СУШКИ МОЛОЧНЫХ ПРОДУКТОВ)[1]

Шиянова Наталья Ивановна, к.т.н., доцент, Филиал Московского государственного университета технологий и управления в г. Мелеузе, Россия, Мелеуз, shianova58@mail.ru
Каяшев Александр Игнатьевич, д.т.н., професор, Уфимский государственный нефтяной технический университет, Россия, Стерлитамак, Kayashev.ai@rambler.ru
Хардина Алина Евгеньевна, ассистент, Филиал Московского государственного университета технологий и управления в г. Мелеузе, Россия, Мелеуз, HardLinka@mail.ru

[1] Статья рекомендована к опубликованию в журнале "**Автоматизация в промышленности**"

Особую значимость в условиях рыночных отношений приобретает вопрос обеспечения населения высококачественными продуктами питания отечественного производства, в частности, сухими молочными продуктами (сухое цельное молоко, сухое обезжиренное молоко, сухие сливки и т.д.). В связи с этим актуальным является разработка автоматизированных систем ресурсосберегающего управления процессом сушки молока, обеспечивающих повышение качества готовой продукции и ее конкурентоспособность.

Технологические процессы в молочной промышленности характеризуются высокой интенсивностью и сложностью. Сырье для производства сухого молока и молочных продуктов имеет нестабильные качественные и количественные характеристики, что требует контроля и управления параметрами технологического процесса в режиме реального времени.

При разработке автоматизированной системы управления процессом сушки молока и молочных продуктов в распылительных сушильных установках необходимо обеспечить заданную влажность продукта и производительность установки.

В процессе производства сухих молочных продуктов значения параметров технологического процесса могут изменяться, вызывая отклонения от нормального режима (регламента). Поддержание параметров технологического процесса в диапазоне заданных значений выполняет система автоматического регулирования (САР). В нашем случае объектом управления является сушильная установка, которая состоит из сушильной башни и калорифера для подогрева воздуха.

Явление высушивания влажных предметов широко распространено в природе и в быту. Обычным носителем процесса сушки является воздух. Чем выше температура воздуха и его скорость, тем сушка протекает интенсивнее.

В молочной промышленности для сушки применяется чистый воздух. В отличие от выпаривания, которое осуществляется под вакуумом, сушка ведется большей частью при атмосферном давлении. Влага при сушке поглощается и уносится воздухом. Количество поглощенной воздухом влаги зависит от свойств воздуха и его способности растворять водяные пары.

При конвективной сушке воздух не только поглощает и уносит влагу, но и одновременно является источником тепла, которое необходимо для того, чтобы превратить воду в пар. Эта двойственная роль воздуха в качестве влагопоглотителя и теплоносителя предопределяет построение математической модели объекта управления [1].

Для выбора оптимальной структуры неоднородных и нестационарных материальных потоков в технологической системе молочного производства следует использовать комплексную имитационную модель производства, воспроизводящую различные альтернативные варианты на ЭВМ для оценки их оптимальности по выходу и качеству готовой продукции, рациональности использования сырья и оптимизации технологических режимов [3].

Создание имитационной модели производственной деятельности предприятия с воспроизведением различных технологических ситуаций, возникающих в процессе комплексной переработки сырья в заданный ассортимент продукции, базируется на следующих основных принципах:

– адекватное отражение структуры свойств и особенностей комплекса технологических процессов, с возможностью варьирования параметров воспроизводимых процессов в исследуемой области;

– максимальное использование априорной информации об объекте, обеспечивающее возможную на этапе моделирования информационную разгрузку оператора в процессе принятия решения с оптимальным распределением функций между человеком и машиной;

– гибкость и самоорганизация модели с возможностью обновления, дополнения и усложнения выполняемых функций;

– реализация модели на ЭВМ с помощью пакета или системы прикладных программ с многоуровневой, иерархической модульной структурой управления совокупностью модулей,

каждый из которых осуществляет законченное преобразование информации.

Процесс сушки молока и молочных продуктов осуществляется в сушильной установке следующим образом.

Очищенный свежий воздух с температурой t_0 и влагосодержанием d_0 подается вентилятором *1* в калорифер *2*, где при постоянном влагосодержании подогревается до температуры t_1 и затем снизу поступает в сушильную башню *3*, внутри которой сверху на вращающий распылительный диск *4* падает сгущенное молоко. Отработавший воздух выходит из сушильной башни с параметрами t_2 и d_2 [2] Диаграмма варианта использования представлена на рисунке 1 [3].

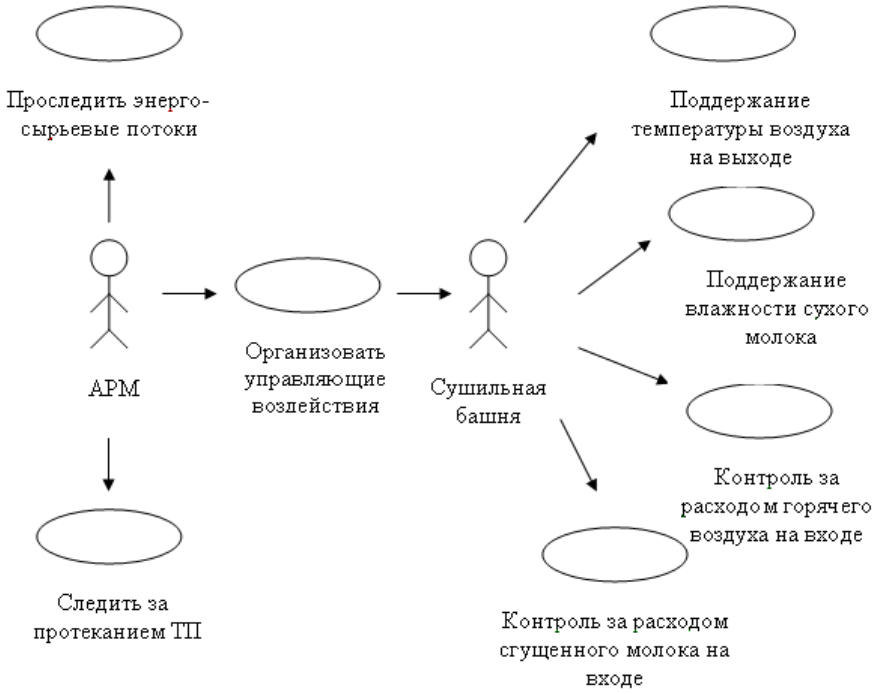


Рис. 1 – Диаграмма варианта использования

Соответствие между основными входными и выходными параметрами и возмущающими воздействиями сушильной установки представлено в виде диаграммы варианта последовательности (рис 2).

Входными параметрами для калориферов являются:

- давление греющего пара на входе в калориферы 1 и 2 – $p_{гр\ пара}'$ и $p_{гр\ пара}''$.

Выходными параметрами для калориферов являются:

- температура горячего воздуха на выходе из калорифера 1 (входе в сушильную башню) – $t_{1в}'$;
- температура горячего воздуха на выходе из калорифера 2 (температура горячего воздуха на входе в сушильную башню) – $t_{1в}''$.

Возмущающими параметрами для калориферов являются:

- температура окружающего (наружного) воздуха – $t_{нв}$;
- расход окружающего воздуха – $Q_{ов}$.

Входными параметрами для сушильной башни являются:

- температура, расход, влажность горячего воздуха на входе в сушильную башню – $t_{1в}'$, – $t_{1в}''$, $Q_{1в}$, $\varphi_{1в}$;
- температура, расход, влажность сгущенного молока на входе в сушильную башню – $t_{1п}$, $Q_{1п}$, $\varphi_{1п}$;
- давление пара на паровую турбину- $p_{пара\ на\ турбину}$.

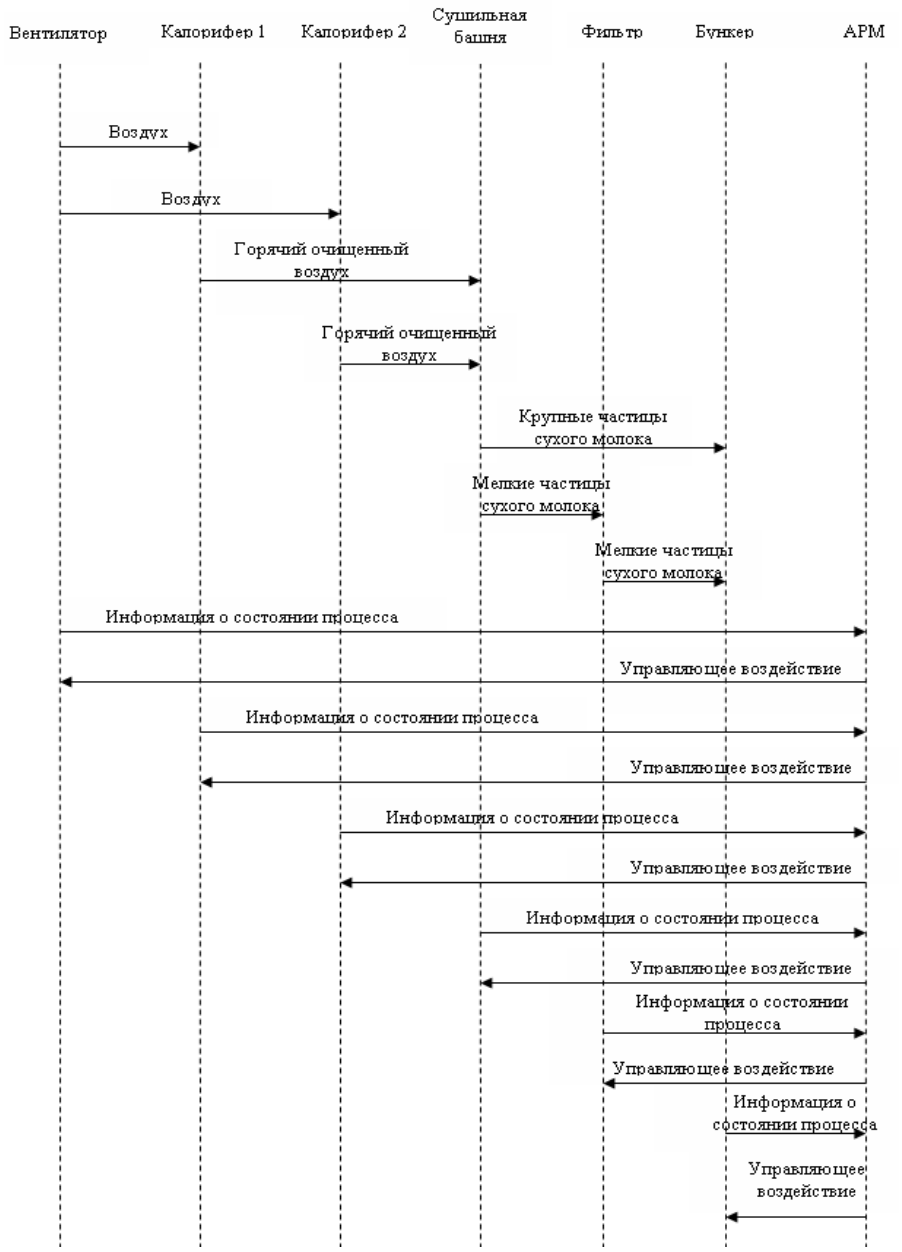


Рис. 2 – Диаграмма последовательности

Выходным параметром для сушильной башни является:

- температура воздуха на выходе из сушильной башни – $t_{2в}$.

На данном объекте управления выделим управляющие и возмущающие воздействия. Управляющими воздействиями могут служить расход сгущенного молока, температура и расход горячего воздуха. На ЗАО "Мелеузовский молочноконсервный комбинат" проведены экспериментальные исследования, управляющим воздействием рассматриваемого объекта регулирования выбрана температура воздуха на входе в сушильную башню.

Остальные параметры считаются возмущающими. Причем природа возмущающих воздействий носит случайный характер, то есть на объект управления интенсивно воздействуют неконтролируемые возмущения, причем они аддитивны.

Поэтому в данном случае целесообразно все неконтролируемые возмущения аддитивно заменить одним эквивалентным вектором возмущающих воздействий.

Анализ объекта управления показал, что для поддержания влажности сухого молока и молочных продуктов в пределах 3-3,2 %, достаточно поддерживать температуру на выходе из сушильной башни на уровне 65-75^{o}C, также необходимо, чтобы влажность сгущенного молока, поступающего в сушильную установку, находилась в пределах 58-52% влаги (42-48% по концентрации сухих веществ), а число оборотов распылительного диска составляло 8

тыс. в минуту. Данный диапазон температур позволит поддерживать система автоматического регулирования влажности готового продукта, в которой применяется цифровой регулятор на базе микропроцессорного контроллера.

Анализ работы сушильной установки показал, что обеспечить статическую нагрузку сушильной башни довольно проблематично, так как поступающее сырье (сгущенное молоко) в сушильную башню неоднородно по количеству и качеству (различной влажности и температуры), а следовательно, в системе периодически возникают переходные процессы. В частности:

- повышение влажности сухого продукта, способствует понижению температуры воздуха на выходе из сушильной башни, и, как следствие, снижение качества готовой продукции и рыночной цены на нее;
- снижение влажности сухого продукта, наоборот, приводит к повышению температуры воздуха на выходе из сушильной башни, и, как следствие, потери готового продукта из–за ее выгорания.

Анализ технологической системы как объекта управления позволил сформировать систему критериев, комплексно характеризующих эффективность технологического процесса сушки молока и молочных продуктов.

Результаты обследований объекта управления – сушильной установки "Нема-500", а также анализ литературных данных показывают, наиболее значимым критерием эффективности технологического процесса сушки молока и молочных продуктов является влажность сухого молока и молочных продуктов как основной показатель качества готового продукта.

В настоящее время существуют приборы для измерения влажности в реальном режиме времени. Однако данные приборы обладают невысокой надежностью, и включение их в контур регулирования может привести к непредсказуемым последствиям, поэтому для управления процессом используют косвенный параметр – температуру воздуха на выходе из сушильной башни.

Причем системой автоматического регулирования (САР) температура воздуха на выходе сушильной башни должна поддерживаться в пределах 65-70^{o}С, тогда влажность готового продукта будет составлять 3-3,2%.

Регулирование температуры воздуха на выходе из сушильной башни изменением расхода сгущенного молока вызывает необходимость автоматического согласования производительностей установок сгущения и сушки.

На реально действующем объекте – сушильной установке "Нема-500" ЗАО "Мелеузовский молочно-консервный комбинат" – получены переходные характеристики объекта регулирования по различным каналам, которые использовались в дальнейшем для настройки параметров типовых регуляторов.

На рис. 3 представлена переходная функция сушильной башни по каналу «расход сгущенного молока – температура воздуха на выходе из сушильной башни» [3].

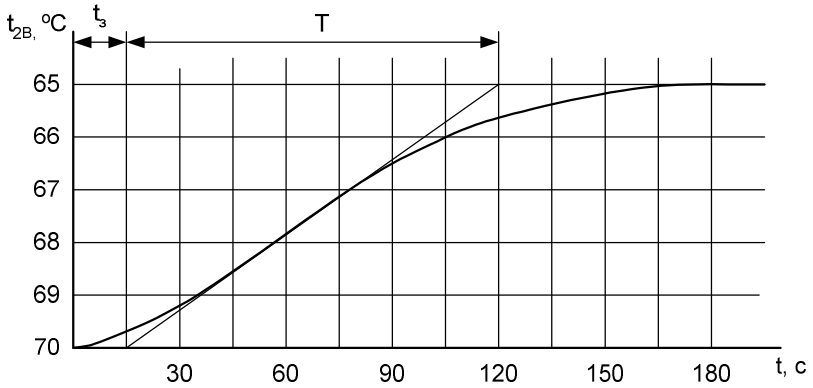


Рис. 3 – Переходная функция сушильной башни по каналу «расход сгущенного молока – температура воздуха на выходе из сушильной башни»

Из графика (рис. 3) видно, что время запаздывания для сушильной башни по данному

каналу составляет $\tau_з$=15 с, постоянная времени T=105 с.

На данном этапе нет практической возможности проводить исследования по каналам "влажность сгущенного молока – температура воздуха на выходе" и "температура сгущенного молока – температура воздуха на выходе", так как воздействовать на параметры влажность и температуру сгущенного молока весьма затруднительно.

Для разработки динамической модели сушильной башни представим уравнение теплового баланса в виде:

$$\frac{dQ(t)}{dt} = Q_{вх}(t) - Q_{вых}(t-\tau), \quad (1)$$

где

$\frac{dQ(t)}{dt} = V\rho c\frac{dT_3(t)}{dt}$ – изменение количества тепла в объеме сушильной башни, причем:

V – объем сушильной башни, м^3;

ρ – плотность смеси внутри сушильной башни, кг/м^3;

c – удельная теплоемкость смеси внутри сушильной башни, Дж/кг;

τ – постоянная времени транспортного запаздывания, в секундах (с).

Раскрывая правую часть уравнения, получим дифференциальное уравнение первого порядка:

$$V\rho c\frac{dT_3(t)}{dt} + G_k c_3 T_3(t) = L(I_1 - I_2) + G_H c_2 T_2(t-\tau). \quad (2)$$

Обозначив $V\rho c$ через k_1, $G_k c_3 - k_2$, $G_H c_2 - k_3$, $L(I_1\text{-}I_2) - k_0$, представим уравнение (2) в виде:

$$k_1\frac{dT_3(t)}{dt} + k_2 T_3(t) = k_0 + k_3 T_2(t-\tau). \quad (3)$$

Уравнение (3) представляет собой динамическую модель сушильной башни.

Приведем уравнение (3) к стандартному виду, для чего воспользуемся следующими обозначениями:

$$T = k_1 / k_2;\ K_{об} = k_3 / k_2; \quad (4)$$

$$y(t) = T_3(t) - k_0 / k_2;\ u(t-\tau) = T_2(t-\tau). \quad (5)$$

С учетом обозначений (4) и (5) уравнение объекта (3) принимает вид:

$$T\frac{dy(t)}{dt} + y(t) = K_{об}u(t-\tau). \quad (6)$$

На основании уравнения (6), устанавливающего взаимосвязь между входным u(t) и выходным y(t) сигналами объекта, получим выражение передаточной функции объекта (сушильной башни) по каналу управления:

$$W_{об}(s) = K_{об}\frac{e^{-\tau s}}{1+Ts}. \quad (7)$$

С использованием математической модели объекта были рассчитаны параметры настройки типовых регуляторов и проведены исследования имитационных моделей системы контроля и управления процессом сушки молока с использованием SCADA-системы Trace Mode.

Литература

1. Бредихин С.А., Космодемьянский Ю.В., Юрин В.Н. Технология и техника переработки молока. М.: Колос, 2003. 400 с.
2. Брусиловский Л.П., Вайнберг А.Я. Автоматизация технологических процессов производства молочных консервов. – М.: Пищевая промышленность, 1975. – 280 с.
3. Технология концептуального проектирования/ Под ред. С.Н. Никанорова. 2-е стереотип. изд., – М.: Концепт. – 2008. – 580 с.
4. Ивашкин Ю.А., и др. Моделирование производственных процессов мясной и молочной промышленности. – М.: ВО «Агропромиздат», 1987. – 232 с.: ил.

УДК 004.588 + 004.023

МЕТОД ПРЕДСТАВЛЕНИЯ ЗАДАНИЙ В ОБУЧАЮЩЕЙ СРЕДЕ ПО ПРОГРАММИРОВАНИЮ

Лаптев Валерий Викторович, к.т.н., доцент, Астраханский государственный технический университет, Россия, Астрахань, laptev@ilabsltd.com

Введение

В работе [1] автором сформулированы требования к среде обучения программированию. В частности, среда должна содержать большой набор однотипных вариантов заданий для написания учебных программ по различным темам. С одной стороны, при обучении выполнение множества однотипных упражнений позволяет добиться прочного усвоения данной конкретной темы. С другой стороны, однотипные задания существенно облегчают преподавателю сравнительную оценку работ обучаемых, поскольку для каждого данного типа задания можно выработать единообразные критерии оценивания.

Однако фиксированный, хотя и достаточно большой, набор подобных заданий делает обучающую систему фактически «одноразовой» – она утрачивает свою обучающую функцию после того, как пользователь выполнит все задания. При «живом» обучении преподаватель, даже имея некоторый небольшой набор однотипных заданий, может «на лету» немного переформулировать задание, изменить значения и/или типы входных данных, изменить формат вывода и т.п. Очевидно, что обучающая система должна обладать аналогичной способностью генерации варианта «на лету».

В статье [2] описан подобный подход в системе обучения программированию на основе параметризуемых тестовых заданий открытого типа. Тестирование можно применять для обучения программированию, но все же основным видом упражнений должны быть задания на программирование.

Еще одним обязательным требованием к среде является обеспечение двух режимов работы системы: обучающего и контролирующего. Для реализации обучающего режима среда должна по каждому типу задания содержать сценарий выполнения. В сценарии в том или ином виде должен быть прописан порядок действий, обеспечивающий правильное выполнение задания. Для реализации контрольного режима в среде должны храниться эталонные результаты выполнения задания. Работа обучаемого, выполненная в контрольном режиме, сравнивается с эталонным решением и соответствующим образом оценивается.

Таким образом, в каждое задание по программированию представляет собой специальный объект в системе, включающий три элемента:

$T = < D, S, M >$,

где D – описание задания, S – сценарий выполнения задания, M – модель эталонного решения задания. При разработке обучающей среды по программированию для удовлетворения указанных требований необходимо решить три задачи:

- разработать механизм генерации вариантов на основе описания D;
- разработать средства представления сценария S и интерпретатор сценариев;
- разработать средства представления модели эталонного решения M и метод оценки работы обучаемого.

В данном докладе предлагается решение проблемы представления задания в едином формализме.

1. Предлагаемое решение описанной проблемы

Решение проблемы представления и генерации вариантов можно осуществить на основе морфологического синтеза. Типовое задание на программирование по некоторой теме нужно преобразовать в параметризованный шаблон, определив и выделив параметры, которые и должны генерироваться системой для каждого конкретного варианта. Различные сочетания параметров представляют разные варианты заданий. Параметры делятся на два

вида: параметры-тексты и параметры-числа. Конкретный вариант параметра-текста всегда выбирается (возможно, случайным образом) из некоторого фиксированного набора значений. Конкретный вариант числового параметра может быть либо выбран из множества значений, либо вычислен с помощью некоторой функции (например, с помощью датчика случайных чисел). На значения параметров (как числовых, так и текстовых) могут быть наложены некоторые ограничения.

Пусть в шаблоне определено **n** параметров и пусть количество значений параметра $\mathbf{p_i}$ равно $\mathbf{m_i}$. Если параметры попарно независимы, то максимальное количество вариантов, которые может сгенерировать система по шаблону, определяется известной формулой:
$N = m_1 \times m_2 \times \ldots \times m_n$

Реально количество вариантов будет немного меньше, поскольку параметры часто зависят друг от друга.

В качестве примера преобразования типового задания в параметризованный шаблон рассмотрим задание из главы о вводе-выводе в сборнике [3]:

> *Написать функцию, которая формирует файл, записывая в него* 100 *чисел в диапазоне от* -50 *до* +50. *Этот файл является входным для другой функции, которая формирует новый выходной файл, преобразуя числа входного файла. Вариант преобразования: добавить к каждому числу последнее число файла. Получение числа, необходимого для преобразования, реализовать в виде отдельной функции.*

Параметризация начинается с того, что вместо конкретных числовых констант задаются числовые параметры K, $x_н$, $x_к$, причем эти параметры должны удовлетворять ограничениям: $x_н \le x_к$, $K > 0$. Можно ограничить K и сверху, например $K < 100$.

Этих параметров недостаточно для генерации набора вариантов, поэтому задание подлежит дальнейшей параметризации. Очевидный параметр – вариант преобразования записей первого файла в записи второго файла. В сборнике [3] приведено 20 вариантов подобных преобразований. Конкретное значение данного параметра представляет собой текст, поэтому такой параметр называется текстовым. При генерации конкретного задания вариант преобразования можно выбирать случайным образом из множества хранимых в системе значений.

Для вышеуказанного задания можно определить еще несколько текстовых параметров:

1. Способ задания K;
2. Способ задания $x_н$, $x_к$;
3. Тип чисел из диапазона $x_н$, $x_к$;
4. Способ задания чисел из диапазона $x_н$, $x_к$;
5. Тип первого файла;
6. Способ задания имени первого файла;
7. Тип второго файла;
8. Способ задания имени второго файла

Например, способ задания K (и способ задания $x_н$, $x_к$) может принимать следующие значения: ввод с клавиатуры; ввод из текстового файла; параметр командной строки; случайное число. Ограничения для K, $x_н$, $x_к$ относятся только к последнему способу задания – генерации случайного числа. Тип чисел может быть целый или вещественный, тип файла – текстовый или двоичный. Имя файла можно задавать самыми разнообразными способами: ввод имени с клавиатуры, как параметр командной строки, – все зависит от фантазии преподавателя. Таким образом, полный список параметров для данного задания будет таков:

1. P_0 : количество чисел K, $K > 0$;
2. P_1 : способ получения K;
3. P_2 : диапазон чисел $[x_н, x_к]$, $x_н \le x_к$;
4. P_3 : тип чисел;
5. P_4 : способ получения $x_н$, $x_к$;
6. P_5 : тип первого файла;

7. P_6 : способ задания имени первого файла;
8. P_7 : тип второго файла;
9. P_8 : способ задания имени второго файла;
10. P_9 : способ обработки значений.

Пусть количество значений каждого параметра равно: $m_1 = 4$, $m_3 = 2$, $m_4 = 4$, $m_5 = 2$, $m_6 = 2$, $m_7 = 2$, $m_8 = 3$; $m_9 = 20$. Для любого набора значений параметров генерируется 1 число K и 1 диапазон чисел $x_н$, $x_к$. Поэтому примем $m_0 = m_2 = 1$. Имя первого файла можно задавать либо посредством ввода с клавиатуры, либо как параметр командной строки. Имя второго файла можно задать, преобразовав имя первого файла, поэтому параметр p_8 равен 3, а не 2. Следовательно, можно сгенерировать вариантов (значение 1 не используется):

$N = 4_1 \times 2_3 \times 4_4 \times 2_5 \times 2_6 \times 2_7 \times 3_8 \times 20_9 = 2^4 \times 4^2 \times 60 = 15360$

Это количество вариантов является теоретическим максимумом, достигаемым при независимости параметров и при отсутствии ограничений. Если потребовать, чтобы для параметров одного вида (p_1 и p_4, p_5 и p_7, p_6 и p_8) генерировались обязательно разные значения, то число вариантов будет меньше:

$N = 4_1 \times 2_3 \times 3_4 \times 2_5 \times 2_6 \times 1_7 \times 2_8 \times 20_9 = 2^4 \times 4 \times 60 = 3840$

Требование, чтобы для параметров одного вида генерировались обязательно разные значения, представляет собой пример ограничения для текстовых параметров. Такие ограничения, как правило, регламентируют допустимые сочетания значений текстовых параметров. Например, ограничение $p_5 = p_7$ означает, что в генерируемом варианте типы первого и второго файла должны совпадать; $p_1 \neq p_4$ – это означает, что в генерируемом варианте способ задания K и способ задания диапазона $x_н$, $x_к$ должны быть различными. Следовательно, множество ограничений можно рассматривать как множество значений специального текстового параметра, который участвует в генерации варианта задания наравне с явно выделенными параметрами. Множество значений этого параметра имеет одно специальное значение – отсутствие всяких ограничений. Это означает, что параметр не участвует в генерации конкретного варианта.

Таким образом, описание задания D в системе представлено в виде шаблона, который включает три элемента: текст типового задания, список параметров и их возможных значений, список ограничений. Для представления шаблона в обучающей системе должен быть определен язык представления, который необходим и для представления сценария S выполнения, и для представления модели эталонного решения M.

Эталонное решение является программой, поэтому языком представления должен быть псевдокод. Однако на псевдокоде невозможно представить законченную программу. Более того, некоторые элементы эталонного решения представляют собой параметры шаблона. Например, в объявлении переменной-файла параметром является тип файла (см. выше параметры p_5 и p_7). Поэтому эталонное решение является шаблоном программы. При генерации конкретного варианта текста задания должен генерироваться и конкретный вариант программы на псевдокоде.

Параметры шаблона программы делятся на два вида: текстовые и программные. Текстовыми параметрами являются такие, множество значений которых одинаково как для шаблона текста, так и для шаблона программы. Например, тип чисел (параметр p_3) может быть представлен одним и тем же словом и в тексте задания, и в программе (вещественный, целый). При генерации конкретного варианта задания в шаблон программы на место текстового параметра просто подставляется конкретное значение.

Представление программных параметров в тексте задания и в программе отличается. Для прояснения дела рассмотрим параметр p_1 – способ задания числа K. В текст конкретного вариант задания вместо p_1 будет подставлено текстовое значение (например, ввод с клавиатуры). В шаблон программы должен быть подставлен фрагмент кода. Таким образом, каждый программный параметр связан с одним из параметров шаблона текста, однако значениями такого параметра являются фрагменты кода. Причем каждому текстовому значению сопоставлен свой кодовый фрагмент.

Заметим, что даже при одинаковых текстовых значениях в разных параметрах кодовые фрагменты могут отличаться. Например, для параметров p_0 и p_7 «ввод с клавиатуры» означает разные программные фрагменты: ввод целого числа и ввод строки.

В качестве примера рассмотрим представление функции, выполняющей создание первого файла из приведенного выше задания. На псевдокоде она может выглядеть так:

```
функция Создать(имя_файла:строка, К:целое, Хн:#p3, Хк:#p3)
  переменная F: файл #p5 выходной;
  переменная i: целая; #p3 xi;
  F.открыть(имя_файла);
  i = 0;
  пока i < K
    #p4                          -- получить xi
    если Хн ≤ xi и xi ≤ Xk
      F.вывод(xi);
      i := i + 1;
    конец;
  конец;
  F.закрыть(имя_файла);
конец Создать;
```

В данном фрагменте эталонной программы параметры помечены префиксом # (решетка). Параметры #p3 и #p5 являются текстовыми. При генерации конкретного варианта и в текст задания, в код эталонной программы на место #p3 будет подставлено текстовое значение «целый» или «вещественный», а на место #p5 – значение «текстовый» или «двоичный». Параметр #p4 является программным, поэтому в шаблон текста задания будет подставлен один из вариантов получения чисел $x_н$, $x_к$. Например, выбран вариант «генерация случайного числа». В программу подставляется соответствующий этому значению фрагмент псевдокода:

```
xi := датчик()/(хк-хн)+хн;
```

Шаблон программы одновременно является и сценарием выполнения задания. В обучающем режиме среда задает последовательность действий, которые обязательно должен выполнить обучаемый, чтобы получить правильное решение. В простейшем случае обучаемому предлагается выполнить ввод операторов программы в том порядке, как они заданы в шаблоне эталонного решения. Причем, при возможных затруднениях при вводе среда может выдать подсказку об очередном элементе оператора.

Порядок действий может быть разным, но все они должны быть обязательно выполнены. Порядок действий задается настройками интерпретатора сценариев. Например, при разработке методом сверху вниз с пошаговым уточнением требуется сначала задать главную функцию и прописать пустые функции-заглушки. Шаблон программы позволяет осуществить подобный сценарий обучения на основе отслеживания уровней вложенности операторов в шаблоне. Сначала предлагается задать операторы нулевой вложенности – это заголовки и операторы окончания функций. Затем для каждой функции, начиная с главной, среда предлагает ввод операторов первого уровня, потом – второго, и т. д.

Таким образом, каждое типовое задание по некоторой теме может быть представлено в обучающей среде в виде параметризованного текста. В качестве примера покажем фрагмент представления приведенного выше задания в xml-подобном виде.

```
<task>                                        -- задание
 <template>                                   -- шаблон задания
  <param>                                     -- параметры
  <p0>количество чисел K;
   <type = number, integer/>                  -- тип параметра
  </p0>
  <p1>способ получения K;                     -- программный параметр
   <type = text, select=value_1+value_2/>     -- тип и множество значений
   <code = value_1/>                          -- программный параметр
  </p1>
 …
```

```
 <p5>тип первого файла;
  <type = text, select=value_3 />          -- тип и множество значений
 </p5>
…
<p9>способ обработки значений;
 <limit>                                   -- ограничения
  #p1 = p4
  #p1!= p4
  …
  #p6 = p8
  #p6!= p8
 </limit>
 <values>                                  -- значения текстовых параметров
  <value_1>                                -- первое множество значений
   #ввод с клавиатуры
   #параметр командной строки
  </value_1>
  <value_2>                                -- второе множество значений
   #генерация случайного числа
  </value_2>
  <value_3>                                -- третье множество значений
   #текстовый
   #двоичный
  </value_3>
  …
  <value_5>                                -- второе множество значений
   #добавить к имени файла цифру 1
  </value_5>
  …
>
 </values>
</param>
<text>                                     -- текст шаблона задания
Написать функцию, которая формирует #p5 файл, записывая в него #p0 чисел
типа #p3 в диапазоне #p2. Вариант получения #p0:#p1; вариант получения
#p2:#p4; вариант получения имени файла:#p6. Этот файл является входным для
другой функции, которая формирует новый выходной #p7 файл, преобразуя
числа входного файла. Вариант получения имени файла:#p8; вариант
преобразования:#p9. Получение числа, необходимого для преобразования,
реализовать в виде отдельной функции.
</text>
</template>
<code>                                     -- шаблон кода
 <value_1>                                 -- фрагменты кода
  <cond: 0 < v < 100/>
  # ввод(v);
  # v = датчик()/100;
 </value_1>
…
<template>                                 -- кодовый шаблон
    функция Создать(имя_файла:строка, К:целое, Хн:#p3, Хк:#p3)
      переменная F: файл #p5 выходной;
    …
    конец Создать;
    функция Дай_Число(имя_файла:строка): #p3;
      переменная in: файл #p5 входной;
      переменная , #p3 xi;
      …
      in.ввод(xi);
      #p4
      …
    конец Дай_Число;
    функция Обработать(имя_файла_1:строка, имя_файла_2:строка)
      переменная in: файл #p5 входной;;
```

```
        переменная out: файл #p7 входной;;
        …
        пока не in.конец()
          in.ввод(xi);
          #p9
          out.вывод(xi);
        конец;
        …
      конец Обработать;
      функция Главная(параметры)
        переменная К: целое; хн, хк: #p3;
        …
        #p6;
        вызвать функцию Создать(имя_файла_1, К, хн, хк);
        #p8;
        вызвать функцию Обработать(имя_файла_1, имя_файла_2);
        выход;
      конец Главная;
  </template>
  </code>
</task>
```

Выводы

Таким образом, представляется вполне возможным реализовать в обучающей системе по программированию генерацию вариантов однотипных заданий на основе представления заданий в виде шаблонов. Это, как было сказано выше, решит проблему создания конкретного варианта задания для каждого студента и повысит устойчивость выработанных навыков написания программ по заданным темам.

Литература

1. Лаптев В.В. Требования к современной обучающей среде по программированию // Объектные системы-2010 (Зимняя сессия): материалы II Международной научно-практической конференции. Россия, Ростов-на-Дону, 10-12 ноября 2010 г. / Под общ. Ред. П.П. Олейника. – Ростов-на-Дону, 2010. – с. 104-110.
2. Sergey Sosnovsky, Olena Shcherbinina, Peter Brusilovsky, Web-based Parameterized Question as a Tool for Learning, in Allison Rossett (ed): Proceedings of E-Learn 2003, Phoenix, Arizona USA, November 7-11, 2003, p. 2151-2154.
3. Лаптев В.В., Морозов А.В., Бокова А.В. C++. Объектно-ориентированное программирование. Задачи и упражнения. – СПб.: Питер, 2007. – 288 с.: ил.

УДК 004.4, 005

XML-СУБД КАК ВОЗМОЖНАЯ ОСНОВА ДЛЯ ОБЪЕКТНЫХ ТЕХНОЛОГИЙ ИС. ТЕХНОЛОГИЯ MULTYF

Ермаков Илья Евгеньевич, преподаватель, Технологический институт им. Н.Н. Поликарпова "Государственный университет – учебно-научно-производственный комплекс", технический директор, НПО "Тесла", Россия, Орёл, ermakov@metasystems.ru

1. Проблемы прирождённых (native) ООСУБД

Несмотря на успешные применения, которые находит каждая из известных ООСУБД, их массовое применение в АСУ и ИС так и не сложилось.

Представляется, что тому причиной слишком прямолинейное понимание и реализация объектности, непосредственный перенос в СУБД представлений, сложившихся в области языков программирования.

Во-первых, плотно сопряжены между собой хранимые данные и программная логика, что ведёт к ориентации на конкретный язык или семейство языков и, кроме того, нагружает

СУБД новыми нехарактерными им функциями – быть платформой выполнения для программных компонентов.

Во-вторых, такая прямолинейная реализация ООП влечёт за собой отсутствие единой статической модели хранимых данных и языка запросов к ним, которые работали бы независимо от объектной инкапсуляции и программной логики классов.

Почему объединение СУБД и платформы выполнения приводит к проблемам? Вероятно, потому, что разработка языка программирования и его платформы, как и разработка СУБД, являются сложными научно-техническими задачами, и выполнить удачно их проектирование и разработку в виде единого целого гораздо сложнее, чем по отдельности – велик риск получить целый набор дефектов и в одной, и в другой составляющей.

2. Объектная ориентированность: базовые качества и их поддержка

Возможный путь развития объектных технологий в области ИС связан с реализацией иного, более базисного содержания объектной ориентированности, чем это обычно практиковалось. Вообще говоря, смысл у ООП один – обеспечение безболезненного развития системы, за счёт того, что она может быть расширена новыми компонентами без переписывания существующих.

Можно назвать только два существенных свойства, которые требуются от объектной технологии:

- Непосредственная поддержка иерархической структуры данных и возможность обеспечения её соответствия структуре информационной модели ИС;
- Безболезненная расширяемость структуры данных. Во-первых, расширяемость данных облегчает внесение изменений в ИС. Во-вторых, она является основой для моделирования иерархии расширения типов информационных объектов («наследование», иерархия «род-вид»). Этим обеспечивается полиморфизм для хранимых данных и передаваемых сообщений. Полиморфизм – это возможность применять с одной и той же функцией ИС как базовую структуру данных, так и расширенные от неё.

В области языков программирования принцип «ООП = модульность + расширяемые типы данных» был в чистом и ясном виде воплощён Н. Виртом в языках Oberon, а затем ИТ-индустрией – в языке Ada-95 [1].

В области СУБД примером успеха такого подхода являются системы MUMPS-семейства с их разрежёнными многомерными массивами (глобалями) и последующая надстройка в СУБД Cache как объектно-ориентированных, так и реляционных, и XML-интерфейсов к многомерному ядру. Cache является одним из немногих примеров массового успеха объектной технологии СУБД, и нам представляется, что этот успех связан именно со следованием идее, что базовое ядро, обеспечивающее расширяемые структуры данных, является основой для практичной объектной ориентированности.

3. Прирождённые XML-СУБД и их перспективы

Неожиданный тезис, который мы предлагаем вниманию читателей, заключается в том, что набирающие в последнее время популярность прирождённые XML-СУБД, на самом деле, могут являться реальной базой для объектно-ориентированной разработки информационных систем. Концептуальным подтверждением тому является, что два названных в предыдущем разделе существенных объектных свойства как раз являются характерными сильными сторонами XML-модели данных. Эмпирически же ободряющим аргументом являются аналогии с MUMPS-технологиями, которые мы не будем здесь перечислять, но они ощущаются на практике – кроме автора данной статьи, их отмечали некоторые разработчики, имеющие опыт использования обоих подходов.

У текстового формата XML есть существенные недостатки, однако полноценные XML-СУБД (примером которых является отечественная «Седна», разработка ИСП РАН[2]) используют XML, фактически, как формат ввода-вывода, а внутреннее хранение

осуществляется иерархическим бинарным образом, в соответствии с моделью XML-Infoset[3].

Разработка ИС на основе XML-СУБД предполагает привычную для реляционных СУБД схему с отдельным слоем программной логики, который взаимодействует с БД посредством языка запросов XQuery. XQuery, по сути, обладает всеми свойствами, привычными разработчикам на SQL (что неудивительно, если учесть наличие в комитете XQuery ведущих разработчиков SQL), но дополняет их новыми, являясь почти полноценным языком программирования, на котором можно разрабатывать хранимые на сервере БД модули.

Нам представляется, что для раскрытия потенциала XML-СУБД необходимо, во-первых, осознанно эксплуатировать их объектно-ориентированные свойства, во-вторых, разрабатывать на их основе мини-технологии для объектно-ориентированного проектирования и реализации информационных систем.

4. MULTYF как пример мини-технологии на базе XML-СУБД

На базе XML-СУБД «Седна» автором статьи разработано и используется несколько мини-технологий объектного характера. Одна из них – информационная технология MULTYF (multi-typed frames), которая предназначена для моделирования неоднородных данных о сложных объектах, их ортогональной классификации и выполнения запросов.

База данных MULTYF состоит из одноуровневого множества фреймов. Каждый фрейм может быть связан с типами, а также может хранить поля со значениями. Типы MULTYF могут образовывать иерархии «тип-подтипы». Фрейм, связанный с подтипом, связан также со всеми базовыми типами. Связь с типами устанавливается наличием внутри фрейма соответствующих типизирующих секций (в случае подтипов – вложенных). Поля данных могут находиться внутри типизирующих секций, если фрейм обладает ими в силу связи с некоторым типом.

```
<frame id="124">
      <type name="T1">
            <field name="a" format="real">0.15</field>
            <type name="Subtype-Of-T1">
                  <field name="b" format="string"></field>
            </type>
      </type>
      <type name="T2">...</type>
</frame>
```

Семантическое моделирование начинают выполнять следующим образом: разрабатываются две независимых иерархии типов – для сущностей предметной области и для сущностей информационной системы.

Например, для ИС сферы торговли автомобильными комплектующими в иерархии типов предметной области будут такие ветви типов, как: *Модель_Авто.Audi.A6*, *Модель_Авто.Audi.A8*, *Производитель_комплектующих.Hamann_Motosport* и т.д. Как видим, типизация предметной области доводится даже до уровня понятий единичного объёма (имён собственных). Соответственно, типизация сущностей ИС будет включать в себя такие ветви типов, как: *ИС.Карточка_Модели*, *ИС.Карточка_Производителя*.

Как правило, фрейм типизируется одним из типов сущностей ИС – и одним или многими типами предметной области, с которыми имеется связь:

```
<frame id="...">
      <type name="Модель_Авто">
            <type name="Audi"><type name="A6"/></type>
      </type>
      <type name="Тип_Кузова">
            <!-- перечисляем все возможные типы для данной модели -->
            <type name="Седан"/>
            <type name="Хэтчбэк"/>
      </type>
```

```
        <type name="ИС">
              <type name="Карточка_Модели">
                    <filed .... >
                    <field ... >
              </type>
        </type>
  </frame>

  <frame id="...">
        <type name="Артикул">
              <type name="A1034"/>
        </type>
        <type name="Товар">
              <type name="Подвеска"><type name="Амортизаторы">...</type>
        </type>
        <type name="Модель_Авто">
              <type name="Audi">
                    <type name="A6">
                    <!-- может быть и для нескольких моделей -->
              </type>
        </type>
        <type name="Производитель_Комплектующих">
              <type name="Hamann_Motosport"/>
        </type>
        <type name="ИС">
              <type name="Карточка_Артикула">
                    <field name="описание">
                          <field name="русский" format="text">
                          ...
              </type>
        </type>
  </frame>
```

Фреймы имеют идентификаторы, однако они используются только в технических целях (например, для удаления фрейма из базы). Все связи между фреймами устанавливаются ассоциативно через типизирующие секции. Например, фрейм артикула A1034 из примера ассоциирован с производителем Hamann_Motosport, производителям в ИС соответствуют фреймы типа ИС.Карточка_Производителя. Таким образом, карточка данного производителя выбирается по паре типов *Производитель_Комплектующих.Hamann_Motosport* и *ИС.Карточка_Производителя*.

Определён специальный XML-язык запросов MULTYF/QL, который позволяет записать образец для отбора фреймов по наличию типизирующих секций и условиям на значения полей. Также имеются специальные унифицированные компоненты интерфейса пользователя для навигации по ортогонально типизированным данным.

Заметим, что, по сути, технология MULTYF является независимой от языка XQuery и XML-СУБД, она всего лишь быстро реализуется на их основе. Модель данных MULTYF является более специализированной и более семантически нагруженной, чем модель XML-Infoset и может быть реализована непосредственно, с использованием XML всего лишь как формата входного-выходного представления данных.

Видимо, путь семантического нагружения и специализации (уменьшения многообразия) модели данных XML-Infoset представляет интерес для развития объектных технологий ИС.

Выводы

В статье выдвинут тезис о возможности примения XML-СУБД в качестве основы для объектно-ориентированной разработки ИС. Предложен путь создании мини-технологий над XML-СУБД и семантического нагружения и специализации модели данных XML-Infoset. Представлен конкретный пример мини-технологии над XML-СУБД – информационная технология типизированных фреймов MULTYF.

Литература

1. Ермаков, И. Е. Объектно-ориентированное программирование: прояснение принципов? / И.Е. Ермаков // Объектные системы – 2010: Материалы I Международной научно-практической конференции. Россия, Ростов-на-Дону, 10-12 мая 2010 г – Ростов-на-Дону, 2010. С. 130-135.
2. Сайт XML-СУБД «Седна» (ИСП РАН) [Электронный ресурс] Режим доступа: http://modis.ispras.ru/sedna/
2. XML Information Set (Second Edition). W3C Recommendation [Электронный ресурс] Режим доступа: http://www.w3.org/TR/xml-infoset/

The Shakhty Institute (Branch) of South-Russian State Technical University (Novocherkassk Polytechnic Institute)

The South-Russian State University of Economics and Service

Alexander Technological Educational Institute of Thessaloniki

The Fourth International Theoretical and Practical Conference

Addition information can be found at www.objectsystems.ru

Object Systems – 2011
(English session)

(with the publication of conference materials)

Information Partners:

Community of System Analysts

The theoretical and applied scientific-technical journal "**Information Technologies**" with a monthly supplement

The monthly scientific-technical and production journal "**Automation in Industry**

The journal "**Information technologies and computer systems**"

10-12 November 2011, Rostov-on-Don, Russia

The conference is devoted to the principles of design, implementation and support of object systems and includes discussion on a wide range of topics. Well-known scientists and major specialists in the field of corporate information systems, representatives of universities and commercial organizations in Russia and abroad (Greece, Poland, Spain) take part in the conference. The conference is correspondence. At the end of the conference a collection of scientific works of authors is published with assigning it ISBN-code, which will be sent to the major research libraries of Russia. The electronic version of the collection is housed in the leading information directories and it is available on the conference website. This website provides an electronic personal certificate confirming participation in the conference. Participation is free.

Conference Committee

1. Nikolay N. Prokopenko, Doctor of Sciences, Professor, Rector, the South Russian State University of Economics and Service, 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. Euclid Keramopoulos, Ph.D., Lecturer, Alexander Technological Educational Institute of Thessaloniki, Greece, Thessaloniki ***(Co-chair of the conference)***
4. Vladimir I. Bozhich, Doctor of Sciences, Professor, Department of Information Systems and Radio Engineering, the South Russian State University of Economics and Service, Russia, Shakhty
5. 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
6. Elvira Yu. Cherkesova, Doctor of Sciences, Professor, Head of the chair of Information Technologies and Management, Shakhty Institute (Branch) of South Russian State Technical University (Novocherkassk Polytechnic Institute), Russia, Shakhty
7. Anatoly A. Mikhailov, Doctor of Sciences, Professor, Department of Automated Control Systems, South Russian State Technical University (Novocherkassk Polytechnic Institute), Russia, Novocherkassk
8. Vyacheslav G. Krawczyk, Ph.D., Associate Professor of Energy and Life Safety, the South Russian State University of Economics and Service, Russia, Shakhty
9. Ignatios Deligiannis, Ph.D., Associate Professor, Head of Department, Department of Information Technology, Alexander Technological Educational Institute of Thessaloniki, Greece, Thessaloniki
10. Panagiotis Sfetsos, Ph.D, Assistant Professor, Department of Information Technology, Alexander Technological Educational Institute of Thessaloniki, Greece, Thessaloniki
11. Apostolis Ambatzoglou, Researcher, Department of Information Technology, Alexander Technological Educational Institute of Thessaloniki, Greece, Thessaloniki

International Program Committee

1. Piotr Habela, Ph.D., Assistant Professor, Polish-Japanese Institute of Information Technology, Poland, Warsaw
2. Erki Eessaar, Ph.D., Associate Professor, Acting Head of the Chair, Faculty of Information Technologies: Department of Informatics, Tallinn University of Technology, Estonia, Tallinn
3. German Viscuso, MSc in Computer Science, Marketing, Versant Corp., Spain, Madrid
4. Sergei D. Kuznetsov, Doctor of Sciences, Professor, Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University, Chief Scientist, Institute for System Programming of Russian Academy of Science, ACM, ACM SIGMOD and IEEE Computer Society Member, Russia, Moscow
5. 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
6. Alexander Yu. Kiryutenko, Ph.D., CIO, Aston, Russia, Rostov-on-Don
7. Edward G. Galiaskarov, Ph.D., Associated Professor, Ivanovo State University of Chemistry and Technology, Russia, Ivanovo
8. Alexander V. Chekiris, Head of department of engineering design and normative-reference information, NIIEVMservice, Belarus, Minsk
9. Irina Yu. Veklenko, Ph.D., System Analyst, Russia, Chernogolovka
10. Victor V. Malyshko, Ph.D., Associated Professor of Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University, Russia, Moscow
11. Ludmila Yu. Zhilyakova, Ph.D., Senior Scientist, Russian Academy of Sciences, Institute of Control Sciences, Russia, Moscow

12. Karina J. Shakhgeldyan, Doctor of Sciences, Head of IT-department, Vladivostok State University of Economics, Russia, Vladivostok
13. Pavel V. Dobryak, Ph.D., Associated Professor, Ural Federal University, Russia, Ekaterinburg
14. Alexander S. Baikin, Lead Systems Analyst, Avtomir, Russia, Moscow
15. Alexey I. Averin, Systems Analyst, Aston, Russia, Rostov-on-Don
16. Valery V. Laptev, Ph.D., Associated Professor of Automated Information Processing and Control System, Astrakhan State Technical University, Russia, Astrakhan
17. Ilya E. Ermakov, Lecturer, Polycarpov Institute of Technology at the State University – Educational-Scientific-Industrial Center, CTO, NPO "Tesla", Russia, Orel
18. Denis Yu. Ivanov, Consultant, IT Consulting, Russia, Saint-Petersburg

The main topics of the conference

1. Graphical notations used in the Object Design of IS
2. Principles of Object Design of Information Systems
3. Object modeling tools
4. The theory of object-oriented programming
5. Methods (patterns) of object-relational mapping
6. The implementation and use of object extensions in relational databases
7. Design, development and implementation of distributed systems
8. Typical implementations of CIS with the use of object technology
9. Principles of organization and implementation of object databases
10. Problems of implementation of object DBMS
11. Simulation modeling of object systems
12. Temporal object systems
13. Problems of study (teaching) of object technology in the university

Important Dates

01.01.2011 – 15.09.2011 – accepting applications for participation in the conference and papers for publication
16.09.2011 – 30.10.2011 – reviewing and editing the submitted papers
01.11.2011 – 09.11.2011 – sending out notices of acceptance reports to the conference
10–12.11.2011 – formation of the Proceedings
20.12.2011 – 31.12.2011 – accommodation on the conference site an electronic layout of Proceedings with assigned ISBN-codes and personal certificates confirming participation in the conference

Addition information can be found at
www.objectsystems.ru

Шахтинский институт (филиал)
государственного образовательного учреждения
высшего профессионального образования
«Южно-Российский государственный технический
университет
(Новочеркасский политехнический институт)»

Южно-Российский государственный университет
экономики и сервиса

V Международная
научно-практическая
конференция

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

ОБЪЕКТНЫЕ СИСТЕМЫ – 2011

(Зимняя сессия)

(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 декабря 2011 г , Россия, Ростов-на-Дону

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 технологии в ВУЗе получает право бесплатной публикации одного доклада сходной тематики на следующей конференции.

Оргкомитет конференции

1. Прокопенко Николай Николаевич, д.т.н., проф., Ректор Южно-Российского государственного университета экономики и сервиса, 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", Южно-Российский государственный университет экономики и сервиса, 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 "Информационные технологии и управление", 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 Владислав Георгиевич, к.т.н., доц., Кафедра "Энергетика и БЖД", Южно-Российский государственный университет экономики и сервиса, 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., 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., кафедра "Экономика и прикладная математика", Педагогический Институт Южного Федерального университета, Россия, Ростов-на-Дону
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, Преподаватель, Технологический институт им. Н.Н. Поликарпова "Государственный университет – учебно-научно-производственный комплекс", Технический директор, НПО "Тесла", 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

01.06.2011 – 15.11.2011 – приём заявок на участие в конференции и докладов к публикации
16.11.2011 – 30.04.2011 – рецензирование и корректировка присланных докладов
01.12.2011 – 09.12.2011 – рассылка уведомлений о принятии докладов на конференцию и приём квитанций об оплате оргвзноса
10–12 декабря 2011 – формирование сборника трудов и сертификатов
20.01.2012 – 31.01.2012 – размещение на сайте конференции электронного макета сборника материалов с присвоенными ISBN-кодами и именных сертификатов, подтверждающих участие в конференции
I – II квартал 2012 г. – рассылка печатной версии сборника материалов авторам, и в научные библиотеки РФ. Рассылка бумажных именных сертификатов. Регистрация электронной версии в специализированных каталогах

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

Для заметок / Notes

Для заметок / Notes

Для заметок / Notes

Отпечатано 10.06.2011

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 **05.06.2011**. Формат А4
Бумага офсетная. Печать цифровая
Усл.печ.л. **13,7**
Тираж 300 экз
Заказ № 735

ISBN 978-5-9902226-5-6