Шахтинский институт (филиал)
Государственного бюджетного образовательного учреждения
высшего профессионального образования
"Южно-Российский государственный технический университет
(Новочеркасский политехнический институт)"

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

ОБЪЕКТНЫЕ СИСТЕМЫ – 2013

Материалы VII Международной научно-практической конференции

Россия, Ростов-на-Дону
10-12 мая 2013 г.

Шахтинский институт (филиал)
Государственного бюджетного образовательного учреждения
высшего профессионального образования
"Южно-Российский государственный технический университет
(Новочеркасский политехнический институт)"

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

ОБЪЕКТНЫЕ СИСТЕМЫ – 2013

Материалы VII Международной научно-практической конференции

Россия, Ростов-на-Дону
10-12 мая 2013 г.

УДК 004 (05)
ББК 32.97-018
О-29

Объектные системы – 2013: материалы VII Международной научно-практической конференции (Ростов-на-Дону, 10-12 мая 2013 г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ (ф) ЮРГТУ (НПИ), 2013. – 118 с.

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 бюджетного образовательного учреждения высшего профессионального образования "Южно-Российский государственный технический университет
(Новочеркасский политехнический институт)"

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

Информационные партнёры

Сообщество системных аналитиков

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

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

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

Оргкомитет конференции

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

Международный программный комитет конференции

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

Рецензенты

Божич Владимир Иванович, Михайлов Анатолий Александрович, Шалыто Анатолий Абрамович, 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 978-5-9903782-4-7

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

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

Object Systems – 2013

Proceedings of the Seventh International Theoretical and Practical Conference

Edited by Pavel P. Oleynik

Rostov-on-Don, Russia
10-12 May 2013

Object Systems – 2013: Proceedings of the Seventh International Theoretical and Practical Conference (Rostov-on-Don, 10-12 May, 2013) / Edited by Pavel P. Oleynik. - Russia, Rostov-on-Don: SI (b) SRSTU (NPI), 2013. – 118 p.

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

Conference Organizers

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

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

Information Partners

Community of System Analysts

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

Journal "**AUTOMATION IN INDUSTRY**"

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

ISBN 978-5-9903782-5-4

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**"

[3] Исследование осуществлено в рамках программы фундаментальных исследований НИУ ВШЭ в 2013 году

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

Содержание / Contents

[1] Работа выполнялась в рамках ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009-2013 по теме «Разработка модели автоматизированной системы интеграции открытых виртуальных лабораторных комплексов» государственного контракта номер 02.740.11. 0658 от 29.03.2010

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

УДК 004.413

ОРГАНИЗАЦИЯ УЧЕБНОГО ПРОЕКТА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Галиаскаров Эдуард Геннадьевич, к.х.н., доцент, доцент, Ивановский государственный химико-технологический университет, Россия, Иваново, galiaskarov@isuct.ru
Агеев Михаил Сергеевич, программист, Творческая студия PROFI, Россия, Иваново, msageev@gmail.com
Умнов Денис Михайлович, руководитель Департамента информационных систем, ООО "Восточный экспресс", Россия, Иваново, umnov@oe-it.ru

Традиционными формами обучения являются лекционные и практические занятия. Все эти виды занятий прекрасно зарекомендовали себя многолетней практикой использования. Но все они ориентированы на индивидуальную подготовку студента. Вместе с тем хорошо известно [1], что современные проблемы разработки программного обеспечения информационных систем лежат в первую очередь в области управления проектами, организации проектной команды, взаимодействия специалистов для достижения поставленных задач и т.д. При индивидуальной форме подготовки смоделировать это практически невозможно. В результате на кафедре возникла идея организации учебного процесса в виде проекта по разработке программного обеспечения информационной системы. Сама по себе эта идея, конечно же, не нова. В определенной степени она лежит на поверхности, но трудно решиться на ее реализацию. Тем не менее, 5 лет назад мы решились на это, постепенно приобрели понимание и опыт того, как следует организовать и осуществить такой проект, и готовы познакомить читателя с результатами наших экспериментов.

Цели и задачи, которые нами ставились перед учебным проектом, постоянно менялись и эволюционировали на протяжении этих пяти лет. Сначала задачи были весьма скромными – например, просто посмотреть, что из этого может получиться. И способы их реализации также были в большей степени традиционны. Студенческая группа делилась на несколько команд по 5-6 человек. Каждой такой команде давалась в разработку относительно независимая часть системы. В начале проекта происходило совместное обсуждение требований, состав будущего решения, определение границ каждой отдельной системы. Далее каждая команда прорабатывала свою подсистему, сворачивая остальные, смежные с ней, подсистемы до уровня «заглушек». Об интеграции этих подсистем в единое решение тогда не помышляли. По окончании работ, в конце семестра, каждая команда оценивалась отдельно, с общим голосованием за лучшую реализацию. В целом полученными результатами мы были довольны, понимая, что движемся в верном направлении. Однако проявились и серьезные недостатки такого подхода. Сложно было обеспечить одинаковый уровень способностей в каждой команде, при этом заранее понять это и устранить перекос на стадии формировании команд трудно. Это приводило к тому, что команды работали неравномерно. Сильная команда вырывается вперед, слабая ожидает результатов другой, чтобы воспользоваться ее решением. Это, пожалуй, было самой главной проблемой. Кроме того, при таком подходе было сложно организовать взаимодействие команд и обеспечить интеграцию подсистем в единое решение.

Поэтому мы практически сразу отказались от этого подхода и решили идти по пути «вся группа делает один проект». Для этого нам необходимо было подобрать задачу, с одной стороны, интересную и сложную, чтобы уверенно занять целую группу в течение семестра, но, с другой стороны – относительно простую, чтобы можно было получить решение, полностью отвечающее основным требованиям, за отведенное в рамках учебной дисциплины время. Идеальным решением было бы привлечение к формулировке темы и выдвижению требований внешнего заказчика, реально заинтересованного в получении конечного продукта – это то, к чему следует стремиться. Мы же поступили проще, воспользовавшись темой, достаточно подробно рассмотренной в книге Карла Вигерса [2]. В этой книге

предлагается задание на разработку web-приложения «Cafeteria Ordering System» для online-заказа блюд сотрудниками некоторой вымышленной компании и управления исполнением этих заказов персоналом кафетерия. В книге представлены проработанные примеры документов о границах и образе решения, спецификаций вариантов использования, списков бизнес-правил и спецификации требований. С образцами этих документов можно ознакомиться на сайте проекта [3]. Таким образом, мы сократили длительность проекта за счет исключения этапов сбора и выявления требований, а также способов реализации проекта.

С осознанием пути развития учебного проекта, пониманием того, каким требованиям должен удовлетворять выбор темы проекта, пришло и понимание того, какие задачи и цели мы преследуем этим проектом:

- приобретение студентами опыта и навыков совместной работы по достижению определенных результатов, а именно, получения достаточно качественно функционирующего программного обеспечения информационной системы;
- приобретение коммуникативных навыков при общении с заказчиком, между собой для достижения поставленных целей;
- знакомство на практике с различными ролями в проекте: аналитика, проектировщика, программиста, дизайнера, тестировщика, руководителя, их функциональными обязанностями и ответственностями;
- знакомство на практике с основами организации и управления проектом, стадиями жизненного цикла системы, дисциплинами проектной деятельности;
- изучение теоретических и практических основ анализа и разработки требований;
- изучение основ web-программирования;
- изучение популярных фреймворков, ускоряющих и стандартизирующих разработку web-приложений;
- знакомство на практике с инструментами совместной разработки, в первую очередь с инструментами управления задачами, контроля версий, моделирования и проектирования систем.

Поставленные цели довольно обширны и достаточно амбициозны. Каким же образом их можно достичь, хватит ли для этого времени и способностей студентов? Что касается возможностей и способностей студентов, то учебный проект мы проводим на 5 курсе в течение осеннего семестра. К этому моменту студенты уже завершили изучение большинства общепрофессиональных и специальных предметов, имеют выполненные курсовые работы и проекты, в том числе по дисциплине «Проектирование информационных систем». Таким образом, студенты обладают достаточными для успешного выполнения учебного проекта профессиональными компетенциям. С другой стороны, учебная дисциплина, в рамках которой проводится учебный проект, включает 112 аудиторных часов, или 6 часов занятий в неделю. С учетом самостоятельной работы этого времени вполне достаточно, чтобы решить поставленные задачи и получить продукт приемлемого качества.

Прежде чем начать выполнение учебного проекта, следует получить согласие студентов на участие в нем. При этом объявляются и тщательно объясняются все правила «игры». Мы применяем следующие правила:

- все студенты освобождаются от экзамена в случае успеха в реализации проекта;
- каждый студент получает оценку в соответствии с «заработанными» баллами;
- часть баллов студент получает по результатам тестирования и контрольных заданий, которые он выполняет после изучения теоретических основ;
- другую часть он получает после оценивания каждой итерации;
- в нашем случае итерация – это одна или две недели, т.е. 3 или 6 занятий, соответственно;
- каждая итерация имеет определенную стоимость;
- студенты представляют результаты итерации, а преподаватель оценивает их,

«выдавая» полную стоимость или даже несколько больше, в случае успешного ее выполнения, или только часть стоимости, если имеются серьезные ошибки или план итерации выполнен не полностью;

- стоимость итераций постепенно повышается к концу проекта;
- на определенной стадии к оценке результатов каждой итерации подключается заказчик или инвестор.

В нашем случае роль инвестора играет практикующий специалист, менеджер проектов, которому отдается 50% всех баллов, которые может получить студент по данной дисциплине[1]. При этом инвестор-заказчик оценивает общее содержание проекта, соответствие демонстрируемого решения его ожиданиям, динамичность развития приложения. Преподаватель, в свою очередь, в большей степени оценивает качество спецификаций, моделей, найденных решений, следование фреймворку, принципам ООП.

После получения согласия начинается формирование команды. На этом этапе происходит деление группы на аналитиков, программистов, тестировщиков. Деление производится на добровольной основе, предоставляя возможность студентам самим решать, какую роль им хотелось бы играть в проекте. Также выбирается руководитель проекта, руководитель аналитиков, руководитель программистов, руководитель тестировщиков, архитектор проекта и web-дизайнер. Если группа не очень велика, то некоторые роли можно/лучше совмещать. При желании вы можете применять Agile-методики при формировании команды или вообще не выделять конкретные роли.

Формирование команды – важный и ответственный этап. Ясно, что он не заканчивается распределением ролей и ответственностей. Процесс этот довольно сложный и требует постоянного и пристального внимания преподавателя. Главное, чтобы это внимание не было слишком назойливым, но нельзя и отпускать студентов в свободное плавание. У нас дисциплину ведут два преподавателя, и мы делим ответственность следующим образом: одни преподаватель занимается организацией команды и проекта в целом, общей постановкой задачи, планированием итераций, курирует вопросы анализа и разработки требований, использования UML в проектировании решения; другой преподаватель занимается вопросами изучения и использования фреймворка, организации и работы с системами контроля версий, средствами разработки, вопросами web-программирования, реализации базы данных и другими техническими вопросами.

В семестре 17 учебных недель. Под выполнение проекта мы планируем 15 недель и две оставляем на форс-мажорные обстоятельства и анализ результатов проекта. Первые несколько итераций – двухнедельные. Во-первых, потому, что они – первые, а во-вторых, включают не только практические действия, но теоретические занятия по вопросам, связанным с выполнением учебного проекта. Через три-четыре итерации, в зависимости от того, насколько быстро ваша команда сумеет выдать практический результат, который можно показать заказчику-инвестору, они становятся однонедельными.

Несколько первых занятий (первые две недели) посвящаются краткому обзору методологий разработки, детальному рассмотрению процесса разработки, основанному на вариантах использования (Use Case Driven Development) [4], объяснению круга обязанностей каждого в проекте, знакомству с документами по теме проекта, обучению работе в системе управления проектами. Команде ставится задача уточнения списка имеющихся вариантов использования, уточнение наименования (Use Case Name) и составление краткого описания каждого варианта использования (Use Case Description). При этом руководитель проекта формирует список задач в системе управления проектом, определяя срок исполнения и закрепляя каждую задачу за определенным исполнителем.

В настоящее время на рынке представлено достаточное количество систем управления

[1] В нашем университете каждая дисциплина оценивается в 100 баллов. До 50 баллов студент может получить во время семестра и еще до 50 баллов во время зачета или экзамена. Для получения допуска к зачетному мероприятию нужно набрать не менее 26 баллов, дисциплина считается пройденной, если студент получит не менее 52 баллов.

проектами. В нашем случае наиболее подходящей является web-ориентированная многопользовательская система с возможностью свободного некоммерческого использования. Таких систем уже значительно меньше, и выбор существенно ограничен. Вначале мы пытались использовать систему devprom [5]. Эта система предоставляется как SaaS, имеется бесплатный тариф. Но система оказалась сложна для использования, сильно ориентирована на Scrum-методологию управления проектами. В конечном итоге, после пары лет работы в devprom, мы перешли на Redmine [6]. Система полностью бесплатная, достаточно простая в использовании, обладает богатыми возможностями настройки, развитой системой отчетности затраченного на проект времени. Нам особенно понравилась возможность создавать свои типы задач и очень гибко настраивать их жизненные циклы под наши нужды и представления.

С третьей недели занятий начинается следующий этап проекта – планирование второй итерации проекта. К этому моменту мы имеем уточненный список вариантов использования. После его обсуждения производится оценка каждого из вариантов использования по критериям важности, частоты использования и сложности реализации. В данном случае каждый участник проекта, в том числе и преподаватель, осуществляют экспертизу, выставляя каждому варианту использования по каждому критерию оценку от 1 до 10. Полученные значения суммируются, и список сортируется по убыванию полученной оценки. В соответствии с рекомендациями методологии для реализации отбираются 15-20% наиболее значимых вариантов использования. Для каждого варианта использования составляется подробная детальная спецификация. Сначала разрабатывается основной сценарий, и выявляются возможные альтернативные потоки. Одновременно аналитик разрабатывает диаграмму VOPC[1] и составляет по этому сценарию раскадровку, визуализируя в самых общих идеях процесс взаимодействия пользователя и системы.

Для составления спецификаций мы используем wiki, который типично встроен в системы управления проектами, или какой-либо текстовый редактор типа Microsoft Word или LibreOffice Writer. В качестве шаблона спецификации используется формат, предложенный в книге [7, стр. 101]. Для построения UML-диаграмм мы используем Visual Paradigm for UML Standart Edition. Следует заметить, что производитель этого инструмента охотно идет на предоставление академической бесплатной лицензии, а сам продукт бурно и активно развивается. Для обеспечения совместной работы над моделью в Visual Paradigm создается проект и размещается в SVN-репозитории, например на http://code.google.com [8].

Одновременно с этим группа программистов-разработчиков под руководством второго преподавателя изучает практику использования выбранного фреймворка разработки web-приложений. Выбор фреймворков крайне широк, и они, как правило, представляют собой те или иные модификации архитектурного стиля MVC. В нашем случае при выборе фреймворка мы руководствовались уровнем личной экспертизы преподавателя в подобных технологиях. Впрочем, при желании или уверенности в силах студентов выбор инструментов реализации можно оставить за ними. Правда, в этом случае вам следует понимать, что, возможно, вам не хватит личной экспертизы, чтобы быстро и грамотно оценить уровень и качество использования выбранного студентами фреймворка. Так что наш совет прост: используйте то, в чем вы сами уверены на 100 %. В качестве такого фреймворка мы используем Kohana [9]. Kohana — это web-фреймворк с открытым кодом, основанный на PHP5 и использующий концепцию HMVC (Hierarchical Model View Controller). Это достаточно простой и безопасный инструмент построения web-приложений. Кроме того, существует развитое русскоговорящее сообщество и значительный объем материалов на русском языке [10, 11].

В конце третьей итерации, т.е. в конце 6 недели, и в случае, если команда работала достаточно успешно, у нас имеются около 20% отработанных спецификаций вариантов использования, подготовлена первая версия модели данных, нарисованы раскадровки

[1] View of Participating Classes (вид классов, задействованных в сценарии варианта использования)

сценариев взаимодействия пользователей с системой, разработан вариант дизайна сайта, программисты уже освоились с выбранным фреймворком, а тестировщики наметили стратегию и план тестирования. Это уже серьезный результат, который можно представить заказчику-инвестору, чтобы уточнить, правильно ли поняты его требования, соответствует ли полученный результат ожиданиям, и продемонстрировать уверенность в том, что проект будет выполнен в срок. С этого момента итерации становятся еженедельными, а их результаты должны предоставляться заказчику-инвестору. Как и каким образом команда представляет результаты каждой итерации в ходе выполнения проекта – дело самой команды. Преподаватель фактически в этот процесс не вмешивается. В конце каждой итерации он лишь просматривает подготовленные презентации вместе со студентами и дает свои рекомендации. Далее руководитель проекта договаривается с заказчиком о месте и времени проведения презентации результатов проекта. У нас практиковалась skype-конференция, во время которой студенты демонстрировали свои достижения и выслушивали замечания заказчика. По окончанию презентации, согласно ранее сделанным договоренностям, заказчик оценивает работу и начисляет на счет команды определенную сумму баллов. У нас стоимость одной итерации на одного человека равнялась 5 баллам, так что при количестве 20 человек в группе команда получала до 100 баллов за итерацию. Причем если заказчик давал за итерацию менее 100 баллов – оставшиеся баллы «сгорали» и уже не могли быть распределены позднее и как-то повлиять на учебный рейтинг участников проекта.

Далее руководитель проекта и его помощники распределяют эти баллы между членами команды – по справедливости и вкладу. Как вы понимаете, это, пожалуй, самый сложный момент в проекте. Тут возможны несколько сценариев поведения. Например, разделить баллы поровну между всеми членами команды. Скорее всего, именно эту стратегию студенты будут использовать в первый момент. Пусть, не мешайте, просто наблюдайте за их решением. Постепенно, особенно после провальных итераций, когда заказчик-инвестор накажет команду меньшим числом баллов, а то и вообще «подарит» ноль баллов, студенты сами поймут, что делить баллы поровну несправедливо, да и невыгодно. Это поощряет бездельников к бездействию, а активных участников демотивирует – зачем что-то делать, когда баллы можно получить и так? В итоге они начнут делить баллы по степени вклада каждого участника в проект. Таким образом, удается добиться высокой объективности оценки. Да и сама оценка в этом случае служит отличным мотивирующим фактором.

К сожалению, в команде всегда присутствует определенный процент бездельников, которые создают только видимость работы. Важно понять, что и с такими людьми следует работать. Следует понимать и донести до студента, играющего роль менеджера проекта, что иногда проблемы возникают не из-за нежелания, а из-за того, что студент не понял или не знает, что от него требуют. С такими студентами нужно просто поговорить, лучше тет-а-тет, и спокойно объяснить, что от него требуется и как это сделать. Другие студенты задачу понимают, но просто не знают, как им сделать то, что от них требуют, а сказать об этом стесняются. В этом случае преподавателю следует помочь, объяснить задачу, показать и рассказать о том, как это делается. Чаще всего после такого общения студент начинает показывать хорошие результаты. Сложнее мотивировать студента, который не может принимать активное участие в проекте. Да, так тоже бывает. Студент работает, у него маленький ребенок, нет нормальных условий. Тем не менее, нужно найти компромисс, четко очертить, какой результат требуется от студента. К сожалению, встречается и такая категория студентов, которая просто ничего не хочет. Они откровенно игнорируют работу в проекте и практически не реагируют на стимулы. Тут два выхода: ничего не предпринимать, и, в конце концов, студент не наберет нужных проходных баллов и не будет допущен к сдаче экзамена. Другой подход – уволить студента из проекта, как мы бы поступили с нерадивым сотрудником. Лучше всего, если такую работу будете выполнять не вы, а руководитель проекта. Например, мы сначала обсуждаем таких студентов с руководителем проекта, объясняем, как он может действовать, и предлагаем ему разобраться с этой ситуацией. Таким

образом, для студента, играющего роль руководителя проекта, это еще и интересная школа менеджерской практики.

В конце последней итерации, за две недели до завершения семестра (хотя это может произойти значительно раньше) происходит завершение проекта и большая презентация его результатов. На презентацию обязательно приглашается заказчик-инвестор, который окончательно оценивает проект. Если результаты превосходят ожидания заказчика, то он может назначить дополнительные бонусные баллы, которые предварительно резервируются. С результатами проекта можно ознакомиться здесь [12].

Последним этапом является подведение итогов. Мы просим каждого участника проекта прислать нам свой отзыв и, конечно, даем свой. Большинство делает это с удовольствием, приветствуют такую форму обучения, предлагают использовать ее шире, вносят замечания по организации и дают интересные предложения по организации и реализации будущих проектов. Ну и, конечно, выставляется окончательная оценка каждому. Напомним, что все баллы студенты распределяют сами и некоторые из них могут получить даже больше, чем нужно. У таких студентов имеется право передать излишки другим, повысив их оценку. Не стоит вмешиваться в их решения по этому поводу. Те же, кто не набрал проходного количества баллов, должны будут выполнить какие-то дополнительные задания и идут на экзамен.

Литература

1. Архипенков С. Лекции по управлению программными проектами. – М.: 2009. – 127 с. URL: http://www.arkhipenkov.ru/resources/sw_project_management.pdf
2. Карл И. Вигерс. Разработка требований к программному обеспечению (2-е издание). Microsoft Press, Русская редакция, 2004. 576 с.
3. Документы. Пользовательская документация // Сайт проекта Cafeteria Ordering System (2012). URL: http://projects.isuct.ru/projects/cos2012/documents (дата обращения: 1.03.2013).
4. D. Rosenberg, M. Stephens. Use Case Driven Object Modeling with UML: Theory and Practice. New York: Apress, 2007. 438 p.
5. DEVPROM: универсальный инструмент управления проектами. URL: http://devprom.ru (дата обращения: 1.03.2013).
6. Redmine is a flexible project management web application. URL: http://redmine.org (дата обращения 1.03.2013).
7. Арлоу Д., Нейштадт И. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование. СПб.: Символ-Плюс, 2007. – 624 с.
8. Файл модели проекта cos-2012. URL: http://cos-model-2012.googlecode.com/svn/trunk/ (дата обращения 1.03.2013).
9. Kohana: The Swift PHP Framework [сайт]. URL: http://kohanaframework.org (дата обращения 1.03.2013).
10. Русская документация Kohana 3 [сайт]. URL: http://kohana3.ru (дата обращения 1.03.2013).
11. Все о фреймворке Kohana [сайт]. URL: http://kohanaframework.su (дата обращения 1.03.2013).
12. Cafeteria Ordering System [сайт]. URL: http://main.isuct.ru/cos2012 (дата обращения 1.03.2013).

УДК 004.04

ИЕРАРХИЯ КЛАССОВ ПРЕДСТАВЛЕНИЯ ВАЛИДАЦИОННЫХ ПРАВИЛ ОБЪЕКТНОЙ СИСТЕМЫ

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

Валидация – это процесс проверки вводимых данных на допустимость и соответствие бизнес-правилам организации. Объемы обрабатываемой информации современного предприятия настолько велики, что пользователю невозможно самостоятельно отслеживать выполняемость подобных правил, поэтому необходим общий механизм, позволяющий

декларировать их и проверять в момент сохранения данных. В данной статье подробно описана иерархия классов представления валидационных правил объектной системы.

В настоящее время наиболее часто используемыми являются объектно-ориентированные языки программирования, популярность которых обусловлена наличием ряда хорошо известных преимуществ [1]. В рамках данной статьи предполагается, что программа создается в среде разработки, метамодель которой подробно описана в работе [2].

Решение любой задачи начинается с формирования критериев оптимальности (КО), которым должна соответствовать полученная реализация. Для рассматриваемой задачи были сформулированы следующие критерии:

1. Разработать механизм валидации значения каждого отдельного атрибута класса, представляющего сущность реального мира.
2. Предусмотреть возможность валидации на уровне классов, предполагающую проверку комбинации атрибутов класса с помощью логического условия.
3. Реализовать возможность валидации атрибутов, тип которых предполагает собой дискретные значения заданного диапазона.
4. Предусмотреть возможность наследования валидационных правил.

Рассмотрим требования каждого критерия более подробно. Разработка программного обеспечения предполагает создание иерархии классов и выделение атрибутов, представляющих сущности реального мира и их связи в соответствии с бизнес-процессами организации. Реализация требования КО1 позволит унифицировано прописывать валидационные правила, задающие ограничения на значения каждого отдельного атрибута классов. Например, при создании нового клиентского заказа, предполагающего продажу товара, необходимо предусмотреть обязательный ввод названия товара и цены. Т.е. необходимо обязать пользователя выбрать продукт из справочника и ввести положительное число, представляющее цену товара. Именно выполнение КО1 позволит реализовать подобные валидационные правила.

Бизнес-правила многих предметных областей гораздо сложнее описанных выше и накладывают ограничения сразу на значения нескольких атрибутов. Именно реализация подобных правил предусмотрена при выполнении КО2. Т.е. задается логическое выражение, накладываемое сразу на несколько атрибутов и предполагающее его проверку при сохранении объекта в базе данных. Примером подобного правила является требование непревышения суммы заказа ста тысяч рублей при оплате наличными.

Многие атрибуты классов позволяют сохранить дискретное значение, попадающее в заданный диапазон. Например, количество техники может быть представлено только целым положительным числом, входящим в определенный диапазон, например, от одного до тысячи. Также документы, созданные в текущем году, должны иметь значения в поле "Дата", попадающие в диапазон дат текущего года. Именно реализация подобных правил предусмотрена в КО3.

Одной из особенностей объектно-ориентированного проектирования/ программирования является наследование, предполагающее, в частности, использование атрибутов базового класса в производном классе [1]. Подобная функциональность необходима и при реализации валидационных правил – это обеспечивается выполнением КО4.

На рисунке 1 в виде диаграммы классов унифицированного языка UML [3] представлена разработанная иерархия валидационных правил.

При проектировании объектной системы принято выделять единый базовый класс, представляющий собой корень иерархии, которым в рассматриваемой диаграмме является абстрактный ValidationRule. Для определения валидационных правил на уровне всего класса выделены абстрактный класс ClassCheckingValidationRule, имеющий наследника InvertableClassCheckingValidationRule, позволяющий инвертировать результат проверки и таким образом упростить реализацию бизнес-правил. В настоящее время для класса

предусмотрена возможность описания только логических условий, которые представляются реализованным классом CriteriaCheckingValidationRule.

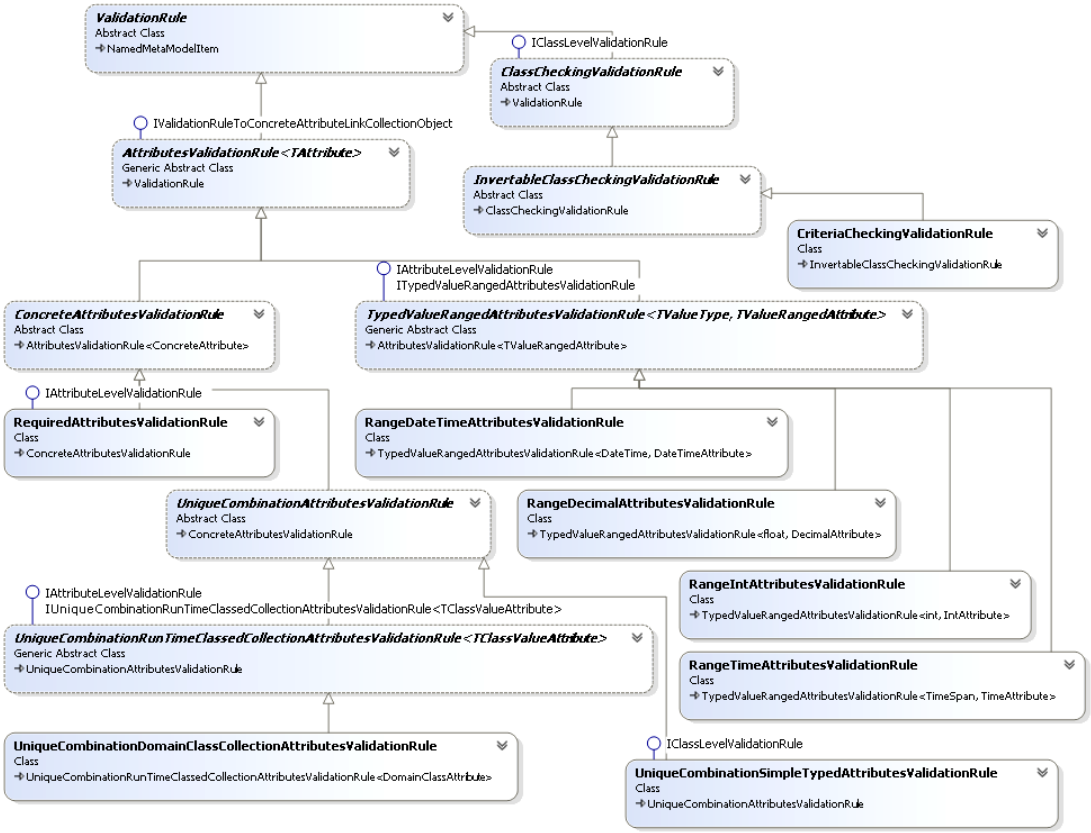


Рис. 1 – Иерархия классов представления валидационных правил

Абстрактный класс AttributesValidationRule позволяет описывать валидационные правила, которые распространяются на отдельные атрибуты. При этом класс является параметрированным, что обусловлено возможностью применения в качестве параметра различных типов атрибутов в производных классах. Унаследованный класс ConcreteAttributesValidationRule поволяет описывать бизнес-правила для атрибутов типа ConcreteAttribute, которые являются основными и добавляются разработчиками. Они подробно описаны в работе [2].

Одним из самых простых правил является необходимость внесения значения в определенные атрибуты класса. Для подобных правил введен реализованный класс RequiredAttributesValidationRule.

Часто необходимо создавать бизнес-правила, требующие наличия уникальной комбинации значений определенных атрибутов. Например, в одно и тоже время не может быть продаж одного и того же товара одному клиенту. Для представления подобных правил введен класс UniqueCombinationAttributesValidationRule, имеющий двух наследников - UniqueCombinationRunTimeClassedCollectionAttributesValidationRule и UniqueCombinationSimpleTypedAttributesValidationRule. Первый класс позволяет указать уникальную комбинацию атрибутов элементов одной коллекции, тип которой создан разработчиком и описывает сущность реального мира прикладной предметной области. В текущей реализации присутствует лишь один наследник UniqueCombinationDomainClassCollectionAttributesValidationRule, параметрированный классом DomainClassAttribute [2].

Второй наследник UniqueCombinationSimpleTypedAttributesValidationRule позволяет создать правило уникальности атрибутов простых типов, таких как целые числа, строки, даты и других типов языка C#, а также атрибутов, значением которых может быть только один (не коллекция) объект предметной области.

Базовый абстрактный класс TypedValueRangedAttributesValidationRule является

корневым для валидационных правил, проверяющих диапазон вводимых значений. Реализованный наследник RangeDateTimeAttributesValidationRule позволяет проверить диапазон введенной даты. Класс RangeIntAttributesValidationRule служит для проверки диапазона целочисленных атрибутов, а RangeDecimalAttributesValidationRule используется для атрибутов дробного значения. Класс RangeTimeAttributesValidationRule предназначен для описания правил, накладываемых на атрибуты, принимающие временные значения.

После описания всех классов рассмотрим соответствие данной иерархии выделенным ранее критериям оптимальности. Система соответствует КО1, т.к. выделен базовый корневой абстрактный класс AttributesValidationRule, позволяющий разработать механизм валидации значения каждого отдельного атрибута класса, представляющего сущность реального мира. При этом унаследованные классы позволяют реализовать требуемое поведение.

Базовый класс ClassCheckingValidationRule представляет возможность валидации на уровне классов и реализовать проверку комбинации атрибутов класса с помощью логического условия, что выполнено в конкретном классе CriteriaCheckingValidationRule. Следовательно, требования КО2 выполнено.

Для реализации требований КО3 для валидации атрибутов, тип которых представляет собой дискретные значения заданного диапазона, введен базовый параметрированный класс TypedValueRangedAttributesValidationRule, имеющий несколько конкретных наследников, позволяющих выполнить проверку атрибутов, сохраняющих значения даты, времени, а также целочисленные и дробные значения.

Требования КО4 выполнены, т.к. при разработке данной иерархии классов представления валидационных правил используется унифицированная метамодель объектной системы, описанная в работе [2] и предполагающая встроенную возможность наследования. Т.к. требования всех критериев выполнены, то можно утверждать, что разработанная система является оптимальной.

В заключение отметим, что дальнейшим развитием описанной метамодели может являться непосредственная реализация описанных правил в графическом интерфейсе пользователя и вывод сообщений об ошибках при их нарушении во время ввода данных конечным пользователем. Также необходимо предусмотреть реализацию механизма описания валидационных правил, вычисляемых динамически с помощью программного кода.

Литература

1. Грэхем И., Объектно-ориентированные методы. Принципы и практика. 3-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 880 с.: ил. – Парал. тит. англ.
2. Олейник П.П. Иерархия классов метамодели объектной системы // Объектные системы – 2012: материалы VI Международной научно-практической конференции (Ростов-на-Дону, 10-12 мая 2012 г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ ЮРГТУ (НПИ), 2012. – С. 37-40
3. Новиков Ф.А., Иванов Д.Ю. Моделирование на UML. Теория, практика, видеокурс. – СПб.: Профессиональная литература, Наука и Техника, 2010. – 640 с.

УДК 004.4’232 +004.588

ИНТЕГРИРОВАННАЯ СРЕДА ДЛЯ ОБУЧЕНИЯ ПРОГРАММИРОВАНИЮ[1]

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

[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

Введение

В работе [1] описан учебный язык программирования Semantic Language и его интерпретатор. Разработка программ на учебном языке должна поддерживаться средой программирования (Integrated Development Environment, IDE), минимальные сведения о которой приведены в той же работе. С одной стороны, эта среда является частью обучающей системы (как указано в работе [2]) и должна поддерживать процесс обучения (например, среда должна обеспечивать преподавателю возможности подготовки обучающих материалов). С другой стороны, интегрированная среда должна быть похожа на современную интегрированную среду для профессиональной разработки программ. Это необходимо, чтобы у обучаемых формировались навыки работы в подобных средах. Поэтому рассмотрим некоторые свойства, присущие современным интегрированным средам.

Промышленная интегрированная среда традиционно объединяет в единую систему множество системных программ: компилятор (и/или интерпретатор), отладчик, мейкер, компоновщик, профайлер и другие. Современная среда в обязательном порядке обеспечивает работу с многомодульными проектами. Однако наиболее важной частью любой интегрированной среды является редактор кода, который и обеспечивает взаимодействие пользователя-программиста со средой.

Как правило, редактор кода представляет собой обычный текстовый редактор, в котором основные операции выполняются с символами, строками и блоками текста. Однако современные промышленные среды обеспечивают некоторую поддержку языка разработки, например: осуществляется цветовая подсветка синтаксиса, которая обычно настраивается в самом редакторе; блоки кода (классы, модули, условные операторы и операторы цикла, определения подпрограмм, и т.п.) разрешается сворачивать и разворачивать. Для ускорения набора кода современные редакторы предоставляют набор так называемых сниппетов (snippets). Сниппет – это короткая комбинация клавиш, с помощью которой в код вставляется конструкция языка программирования; сниппеты, как правило, разрешается настраивать в редакторе.

В последние годы в некоторых средах выполняется интеллектуальный анализ (технология IntelliSense) кода при его создании (до компиляции). Например, при наборе вызова метода некоторого библиотечного класса предоставляется подсказка, позволяющая не набирать конструкцию полностью вручную, а выбрать нужную из списка. Ошибочные конструкции помечаются в окне редактора в момент набора (без компиляции), и в отдельном окне выводятся сообщения о возможных ошибках. Обеспечивается интеллектуальная навигация по коду, например, переход от вызова метода к его определению и обратно. В наиболее развитых средах реализованы некоторые операции рефакторинга [3,4].

Подобные возможности позволяют значительно снизить количество ошибок при наборе кода и существенно повышают производительность программиста. Тем не менее, подавляющее большинство операций, которые программист выполняет в редакторе, – это обычные операции с текстом. Применение текстовых операций к коду программы приводит к регулярному нарушению синтаксиса языковых конструкций. Это, во-первых, «сбивает с толку» интеллектуальный подсказчик, во-вторых, приводит к возникновению мелких синтаксических ошибок, на исправление которых требуется время.

Как правило, редактор интегрированной среды сохраняет код программы в текстовом виде. Текстовое представление кода является входным для компилятора, что приводит к необходимости иметь в компиляторе фазы лексического и синтаксического анализа. С одной стороны, это замедляет процесс разработки программы из-за увеличения времени трансляции. С другой стороны, ошибки компиляции затрудняют процесс обучения, так как отвлекают обучаемого от главной цели – от решения задачи. Кроме того, текст программы можно открыть, просмотреть и изменить вне интегрированной среды. В профессиональной разработке это бывает полезно, но при обучении провоцирует обучаемых на нечестные способы выполнения заданий по программированию.

Б. Страуструп в своей книге [5] отмечал, что главным препятствием на пути развития языка C++ являются символьно-ориентированные инструменты (в частности, текстовый редактор кода). Наиболее перспективный и интересный подход – отказаться от традиционного текстового представления и реализовать инструментарий на основе семантических понятий языка программирования. В этом случае синтаксис языка представляет собой интерфейс между языком программирования и пользователем (программистом). И, как всякий интерфейс, его можно заменять, не изменяя базовой семантики языка.

В соответствии этим подходом и с учетом анализа свойств современных сред авторами был сформулирован ряд концепций реализации обучающей интегрированной среды Semantic IDE, которая поддерживает разработку программ на учебном языке программирования [1].

1. Интегрированная среда Semantic IDE и семантический редактор кода

Внешний вид среды Semantic IDE показан на рисунке 1. Центральное окно – это окно редактора кода. В нем пользователь набирает код программ и пишет текст документов. Для каждого документа создается отдельная вкладка. Над центральным окном расположены главное меню и лента выбранного пункта главного меню.

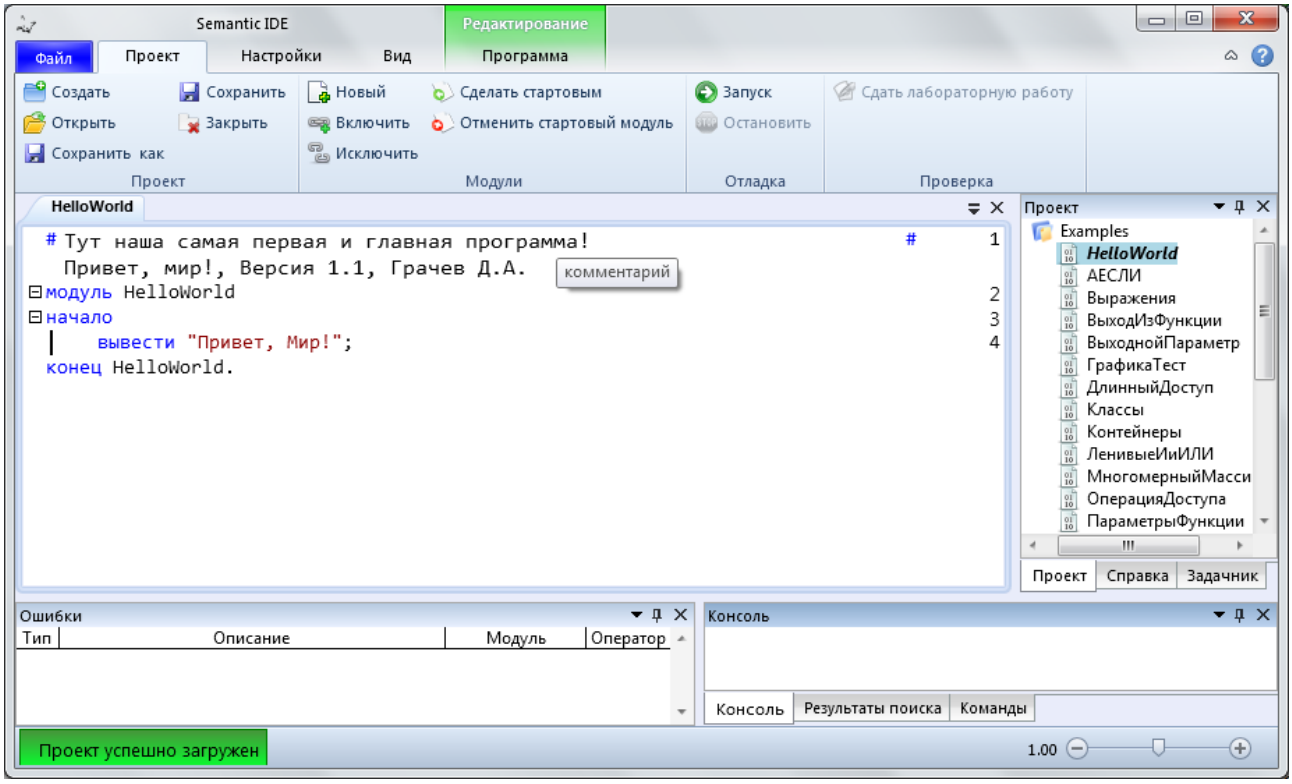


Рис. 1 – Общий вид Semantic IDE

Справа расположено окно проектов. В нем отображается текущий проект, с которым работает пользователь в данный момент. Внизу – окно сообщений об ошибках и окно консоли. Сообщения об ошибках появляются при редактировании кода – для этого не требуется запускать программу на трансляцию.

При выполнении программы среда переключается в окно консоли, в котором программист задает входные данные и в которое выводятся результаты работы программы. При работе в среде не открывается никаких дополнительных окон от операционной системы. В консольном окне (вкладка Результаты поиска) показываются результаты операций поиска, выполняемых пользователем. Например, требуется найти все вызовы конкретной функции, или от вызова функции перейти к определению, чтобы уточнить список параметров и их типы. Во вкладке Команды консольного окна отображаются все действия, выполняемые пользователем в среде.

Все окна, кроме центрального окна редактора, являются плавающими, и их можно перемещать и закреплять как вкладки в центральном окне.

1.1. Проекты

Работа в интегрированной среде начинается с создания проекта. Проект – это набор файлов, созданных в редакторе кода и сохраненных на диск. При создании проекта на диске создается папка, в которую записывается файл проекта. В этой же папке хранятся все остальные файлы, включенные в проект. Проект создается даже в том случае, если включает единственный файл.

Любой файл, созданный в редакторе кода, может содержать программный модуль на учебном языке. Но в общем случае это не обязательно – файл может содержать только неисполняемую информацию. Таким образом, в редакторе кода можно создавать как программные, так и информационные проекты. Примером информационных проектов в Semantic IDE являются проект Справка и проект Задачник, содержащий набор лабораторных работ по программированию.

Программные проекты могут быть исполняемыми и неисполняемыми. Примером неисполняемого проекта в среде является проект Framework, в котором собраны модули стандартной библиотеки. Если же программный проект требуется выполнять, то один из модулей проекта назначается стартовым: выполнение программы начнется с секции инициализации этого модуля. Остальные модули загружаются и связываются по мере необходимости. Примером исполняемого проекта, предоставляемого в составе Semantic IDE, является проект Example, который содержит множество примеров программ на учебном языке (см. рис. 1).

Для хранения проектов на диске был разработан специальный xml-формат. Например, текущая версия проекта Задачник выглядит следующим образом:

```
<?xml version="1.0" encoding="utf-16"?>
  <Project Name="Задачник" StartupModule="#notset">
    <Files>
        <Module Path="Лабораторная_1.sl" />
        <Module Path="Лабораторная_2.sl" />
        …
        <Module Path="Лабораторная_10.sl" />
        <Module Path="Лабораторная_11.sl" />
    </Files>
  </Project>
```

Файл проекта сохраняется в папке проекта в виде xml-файла с расширением «prj», а файлы, созданные в редакторе и включенные в проект, имеют расширение «sl». В данном случае отсутствует стартовый модуль, поскольку проект не является исполняемым.

1.2. Редактирование кода

Редактор кода – это не традиционный текстовый редактор. В Semantic IDE реализован семантический редактор. Как было описано в работе [1], семантический редактор оперирует не символами, а операторами учебного языка и объектами программы. Во-первых, оператор всегда вставляется в код в синтаксически правильном виде. Во-вторых, редактор сам следит за отступами вложенных операторов и не позволяет нарушать структурный вид программы. В-третьих, изменение ключевых слов запрещено, что полностью исключает многие синтаксические ошибки, возникающие из-за ошибок набора ключевых слов. Редактор разрешает символьный ввод только в строго определенных позициях оператора. Например, в операторе вызова разрешено вводить посимвольно имя вызываемой процедуры.

Операторы учебного языка добавляются в код программы с помощью сниппетов и контекстного подсказчика. Каждый оператор учебного языка начинается ключевым словом, поэтому сниппет – это первые две-три буквы ключевого слова оператора. При наборе букв сниппета в кодовом окне редактора подсказчик выводит список операторов и подсветка

устанавливается на конкретном операторе.

Контекстный подсказчик работает и при наборе составного имени при вводе селектора-точки. Если перед точкой было набрано имя модуля, то в списке выводится список всех открытых объектов этого модуля. Если перед точкой было набрано имя объекта определяемого типа, то подсказчик выводит имена всех открытых полей и методов, определенных в этом типе.

Редактор следит за действиями программиста и сообщает об ошибках в момент набора программы. Пока ввод оператора не завершен, в окне ошибок «вывешены» все сообщения об ошибках, которые возникают по мере ввода составных частей оператора. Например, при ошибке в написании имени переменной во время ввода оператора мгновенно появляется сообщение о том, что данная переменная не была определена. Отметим, что в кодовом окне нумеруются операторы программы, а не строки текста (см. рис. 1).

Аналогично выполняется удаление – удаляется вся конструкция целиком. Если же оператор не удаляется, то разрешается замена элементов оператора. При замене тоже работает контекстный подсказчик. Например, в операторе объявления переменной можно заменить тип переменной, выбрав его из списка предложенных типов. Точно так же предлагается список видимых в данной точке имен переменных, если программисту потребовалось заменить имя переменной.

Заметим, что программа при вводе/удалении операторов всегда является синтаксически правильной – синтаксические ошибки отсутствуют. Даже при наборе арифметических или логических выражений всегда вставляется (и удаляется) синтаксически правильная часть выражения. Все возникающие ошибки являются исключительно лексическими (например, неверно набранная числовая константа) или семантическими (например, в операторе присваивания требуется недопустимое преобразование типа). Таким образом, набор кода программы требует минимального количества действий от программиста, и при этом существенно сокращается количество ошибок.

Вместо текстовых операций редактор обеспечивает набор семантических операций модификации кода, которые не нарушают синтаксической корректности. Примером подобной операции может служить вставка объемлющего оператора цикла, преобразующая выделенную последовательность операторов в тело этого цикла. Как обычно, доступные операции можно выбирать из списка, предоставляемого контекстным подсказчиком.

2. Синтаксис как интерфейс

Как сказано в работе [1], редактор кода в соответствии с действиями программиста строит внутреннее представление программы. Один модуль – это дерево с вершинами сложной структуры, которое мы назвали семантическим. Каждая вершина (узел) дерева представляет отдельный оператор программы, в котором сохраняется полная информация о семантике оператора. Например, для оператора присваивания хранится информация о вычисляемом выражении и о переменной, которой присваивается значение выражения.

В каждом узле имеется ссылка на следующий узел-оператор. Таким образом, последовательность операторов представляет собой последовательный список узлов семантического дерева. В узлах, соответствующих блочным операторам учебного языка [1], имеется еще одна ссылка – на первый вложенный оператор тела.

Внутреннее представление программы в виде семантического дерева является входным для интерпретатора. Поскольку все ошибки выявляются при создании кода в редакторе, на вход интерпретатора поступает правильная программа. Интерпретатор не выполняет ни лексического, ни синтаксического анализа – это существенным образом упростило реализацию и повысило быстродействие.

С другой стороны, представление программы в виде семантического дерева позволяет реализовать идею Б. Страуструпа о том, что синтаксис языка программирования является интерфейсом [5], который можно менять, выбирая наиболее подходящий для обучения. В Semantic IDE, помимо синтаксиса учебного языка Semantic Language, реализованы Си-

подобный и Pascal-подобный синтаксисы. Си-подобный синтаксис разработан на основе языка Java [7], а за образец Pascal-подобного принят синтаксис языка Component Pascal [6], реализованного в системе BlackBox Component Builder. Каждое из представлений может быть показано и в русской лексике, и в английской.

Представление программы вычисления факториала на языке Semantic Language в английской и русской лексике показано на рис. 2.

```
module Факториал
start
    variable integer i := 1;
    variable real current := 1;
    constant integer N = 15;
    while i < N do
      let current := current * i;
      let i := i + 1;
      output '\n';
      output current;
    end of while;
end Факториал.
```

```
модуль Факториал
начало
    переменная целое i := 1;
    переменная вещ current := 1;
    константа целое N = 15;
    пока i < N повторять
        присвоить current := current * i;
        присвоить i := i + 1;
        вывести '\n';
        вывести current;
    конец цикла;
конец Факториал.
```

Рис. 2 – Представление программы на языке Semantic Language

Та же программа в Си-подобном и Pascal-подобном виде представлена на рис. 3.

```
package Факториал;
function Main() {
    var int i = 1;
    var double current = 1;
    const int N = 15;
    while (i < N) {
        let current = current * i;
        let i = i + 1;
        Console.Write('\n');
        Console.Write(current);
    } // конец Main
} // конец Факториал
```

```
MODULE Факториал;
BEGIN
    VAR i: INTEGER := 1;
    VAR current: REAL := 1;
    CONST N: INTEGER = 15;
    WHILE i < N DO
        LET current := current * i;
        LET i := i + 1;
        Log.Out('\n');
        Log.Out(current);
    END
END Факториал.
```

Рис. 3 – Представление программы вычисления факториала

а) Си-подобный синтаксис б) Pascal-подобный синтаксис

В коде программы подчеркнуты слова, не входящие в язык представления. Эти слова выводятся в кодовое окно только для повышения читабельности кода и не представлены в семантическом дереве.

Каждый синтаксис определяется собственной грамматикой представления, в которой каждому оператору соответствует правило представления, задаваемое в формате РБНФ. Правила представлены в русской лексике – механизм переключения лексики ключевых слов реализован независимо от грамматики. Например, оператор присваивания в грамматике представления Semantic Language, задан таким правилом:

```
<Assign> ::= "присвоить" <expression> ":=" <expression>";"
```

Нетерминал в левой части правила – это семантическая единица языка (оператор), которая в семантическом дереве представлена соответствующим узлом. В правой части правила – синтаксис представления этого оператора в окне редактора. Последовательности символов, заключенные в кавычки, выводятся буквально в окно редактора – это ключевые слова языка. Нетерминалы в правой части правил – это параметры данной семантической единицы, о них «знает» интерпретатор. В данном случае узел имеет два параметра: слева параметр-выражение, представляющий объект, которому присваивается значение правого параметра-выражения.

То же правило в грамматике Pascal-подобного представления выглядит так:

```
<Assign> ::= "%ПРИСВОИТЬ""% "<expression> ":=" <expression>";"
```

В грамматике Си-подобного представления это правило выглядит следующим образом:

```
<Assign> ::= "%присвоить""% "<expression> "=" <expression>";"
```

Знак процента «%» в начале слова означает, что это слово не входит в состав слов, синтаксис которого определяет грамматика. В данном случае это правило говорит о том, что в Си-подобных и Pascal-подобных языках в операторе присваивания отсутствует ключевое слово «присвоить» (английское «let»). Слово-исключение показано в коде подчеркнутым.

Представление одной и той же программы в разных видах позволяет преподавателю сосредоточить внимание обучаемого на семантике конструкций, а не на синтаксическом представлении.

3. Поддержка разработки обучающих материалов

Одним из операторов, обычно размещаемых в коде, является оператор-комментарий, обозначаемый в окне редактора символом «решетка» (#). Оператор-комментарий может быть вставлен в код в любом месте, где допускается вставка оператора языка программирования, а также перед кодом и после него. Однако в семантическом редакторе разрешается создавать произвольную последовательность операторов-комментариев вообще без программного кода. Редактор вставляет в семантическое дерево отдельный узел для каждого оператора-комментария. Интерпретатор подобные узлы не обрабатывает.

В пределах комментария разрешается посимвольный ввод любого текста, и выполняются традиционные текстовые операции с символами и строками. Разрешены и обычные операции с буфером обмена – это позволяет вставлять фрагменты материалов из других программ (например, из MS Word). В операторе-комментарии можно создать таблицы, разрешается вставлять рисунки и видеоролики. Обеспечивается возможность связывания комментариев с помощью гиперссылок, в том числе и с другими файлами проекта.

Таким образом, возможности представления информации практически не уступают возможностям подготовки документа в MS Word в rtf-формате. Все это позволяет преподавателю непосредственно в редакторе готовить обучающие материалы. Проект Справка и проект Задачник подготовлены именно таким образом. Кроме того, теоретический материал можно проиллюстрировать кодом программы-примера, которую можно запустить на выполнение. Это позволяет демонстрировать образцы хорошего стиля программирования и способствует более быстрому и прочному усвоению изучаемой темы.

Заключение

Реализованная среда Semantic IDE является основой обучающей системы по программированию, которая разрабатывается на кафедре АСОИУ Астраханского государственного технического университета. В настоящее время среда используется для выполнения лабораторных работ по дисциплине «Основы алгоритмизации» и на факультативных занятиях по программированию в Астраханском колледже вычислительной техники. Опыт применения среды показывает существенное сокращение времени при создании кода программы и практически полное исчезновение ошибок набора. По отзывам преподавателей, представление программы в русской лексике облегчает усвоение базовых понятий студентам, не изучавшим основы программирования в школе, и способствует усвоению профессиональной терминологии. С другой стороны, возможность переключить программу в английскую лексику облегчает студентам переход на профессиональные англоязычные языки программирования.

В настоящий момент развитие среды продолжается. Уже разработаны Python-подобный и Basic-подобный синтаксисы представления, что повышает универсальность среды как инструмента обучения. Разрабатывается грамматика представления программы в синтаксисе C#. Исследуются возможности трансляции семантического дерева программы в код на языках, в которых среда может показывать программу. Это позволит переносить программы из Semantic IDE в профессиональные среды.

Важнейшей особенностью Semantic IDE является способность отслеживать любые операции пользователя: от создания проекта до ввода имени в выражении. Действия пользователя представляют собой последовательность операций семантического редактора, которую среда автоматически сохраняет. Во-первых, это позволяет собирать статистику как по операциям, так и по времени их выполнения. Например, среда может отследить, сколько раз в процессе выполнения задания программист обращался к справочным материалам, какие темы просматривал и сколько времени он на это затратил. На основании собранных данных можно выяснять, при изучении какой тематики пользователь испытывает трудности, и соответствующим образом корректировать процесс обучения.

Во-вторых, запись действий позволяет преподавателю при создании обучающих материалов создать типовой сценарий выполнения того или иного задания. Тогда при выполнении студентом конкретного задания появляется возможность сравнивать его действия с типовым сценарием. Если студент совершает слишком много ошибок, то среда сможет не только подсказать, что надо делать, но даже и заставить его выполнить правильные действия, чтобы завершить задание. Подобный режим работы называется институциональным управлением обучением [8].

Другое направление развития – это разработка методов адаптивного управления обучением. На базе интегрированной среды разрабатывается система контроля, которая не только определяет правильность программы, но и оценивает качество созданного кода.

Работа поддержана грантами «У.М.Н.И.К.» в 2011 и 2012 годах, и грантом «Старт» в 2013 году. Работающую версию среды можно загрузить с сайта sem_tech.net.

Литература

1. Грачёв А.Д., Лаптев В.В. Разработка учебного языка программирования и интерпретатора для обучающей среды // Объектные системы-2012: материалы VI Международной научно-практической конференции (Ростов-на-Дону, 10-12 мая 2012г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ ЮРГТУ (НПИ), 2012. – 110 с., с. 92-101.
2. 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. – 134 с., с. 104-110.
3. Пауэрс, Л. Microsoft Visual Studio 2008 / Л. Пауэрс, М. Снелл: Пер. с англ. – СПб.: БХВ-Петербург, 2009. – 1200 с.
4. Давыдов С.В., Ефимов А.А. IntelliJ IDEA. Профессиональное программирование на Java. – СПб.: БХВ-Петербург, 2005. – 800 с.
5. Страуструп Б. Дизайн и эволюция C++. – М.: ДМК Пресс; СПб.: Питер, 2006. – 448 с.
6. Потопахин В.В. Современное программирование с нуля! – М.: ДМК Пресс, 2010. – 240 с.
7. Хорстман К.С., Корнелл Г. Java 2. Библиотека профессионала, том 1. Основы. – М.: ООО «И.Д. Вильямс», 2011. – 816 с.
8. Лаптев В.В. Метод оценивания умений и навыков при обучении программированию // Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика. Научный журнал, № 2 / 2013. – Астрахань: Издательство АГТУ, 2013 г.– 218 с., с.194-201

УДК 004.02

МОДЕЛИРОВАНИЕ СИСТЕМЫ ФОРМИРОВАНИЯ ЗАКАЗОВ И УЧЕТА ТОВАРОВ

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
Михайлюк Екатерина Андреевна, старший преподаватель, Старооскольский технологический институт, Россия, Старый Оскол

Развитие информационных технологий в настоящее время происходит очень динамично. Практически для любой области бизнеса, финансовой и хозяйственной деятельности имеется специальное программное обеспечение, автоматизирующее и упрощающее работу предприятий.

В современных условиях на крупных предприятиях сотрудникам приходится иметь дело с большим количеством часто изменяющейся информации, которую просто невозможно обработать «вручную». На предприятиях, имеющих значительный оборот продукции, существует необходимость учёта и контроля большого объёма кадровой, финансовой, закупочно-сбытовой, производственной, маркетинговой и другой информации.

Актуальность темы данного исследования связана с необходимостью автоматизации процесса формирования заказов и учета товаров в региональных представительствах многофилиального предприятия, занимающегося оптовой торговлей.

ЗАО “АЭРОБЕЛ” - Белгородский завод по производству стеновых блоков из ячеистого бетона автоклавного твердения - объединяет 4 района: Алексеевский, Курский, Воронежский и Старооскольский.

Основными видами деятельности предприятия являются:

- производство изделий из бетона, гипса и цемента;
- производство изделий из бетона для использования в строительстве, включая производство шпал, столбов, труб;
- оптовая торговля материалами, лесоматериалами, строительными материалами и санитарно-техническим оборудованием;
- розничная торговля строительными материалами, не включенными в другие группировки.

На предприятии все сферы процесса производства и функционирования автоматизированы, но существует проблема формирования заказов и учета товаров во всех региональных представительствах.

Основной целью данного исследования является моделирование процесса формирования заказов и учета товаров в региональных представительствах.

Для осуществления поставленной цели необходимо решить следующие задачи:

- связать информационные потоки главного офиса и пяти отдельных региональных представительств в единую информационную сеть по технологии «клиент – сервер»;
- автоматизировать систему формирования заказов и учета товаров;
- создать базы данных с разграничением прав доступа в региональных представительствах и в главном офисе;
- автоматически отправлять данные о реализации товара в конце каждой смены в главный офис;
- разработать модель управления запасами.

При разработке информационной системы необходимо учитывать следующие возможные проблемы:

1. **Потеря связи между базами данных** региональных представительств и главного управления. В процессе передачи информации с региональных представительств в главное управление может быть потеряно соединение (неполадки в сети, плохое соединение и др.), тогда в главное управление поступят некорректные данные.
2. **Оперативное обновление цен**. В процессе работы операторы на региональных представительствах, невовремя получив информацию об изменении цены, могут продавать товары по «старым» расценкам, что приводит к фальсификации отчетности.[1]

Сбор информации для формирования отчетов осуществляется из документов, которые формируются в процессе работы региональных представительств: приходные накладные; продажи; оплаты контрагентов.

Для бесперебойной работы ЗАО «Аэробел» необходима четко налаженная система управления запасами, чтобы поставки происходили вовремя, наиболее экономичными партиями и не создавалось дефицита товара.

Материальные запасы – это запасы, находящиеся на разных стадиях производства и обращения продукции производственно–технического назначения.

Запасы – это необходимая составляющая материального потока, которая существует на протяжении большей части времени его движения.

Предприятия должны поддерживать, возможно, более низкий уровень запасов товаров установленной номенклатуры и при этом условии сохранять высокий уровень обслуживания и оптимальное время поставки.

Процесс управления запасами заключается в решении двух основных задач:

- определение размера необходимого запаса, то есть нормы запаса;
- создание системы контроля за фактическим размером запаса и своевременным его пополнением в соответствии с установленной нормой.[2]

Рассмотрим проблемы управления запасами, связанные с заказом на партию товара поставщику.

В любой системе управления запасами уровень последних изменяется в соответствии с циклической моделью. Процесс снижения уровня запасов определяется соответствующей моделью спроса. В некоторой точке для пополнения запаса будет сделан новый заказ. По прошествии некоторого времени, называемого временем поставки, заказ будет получен и уровень запасов возрастает. После этого начинается новый цикл запасов (рис. 1).

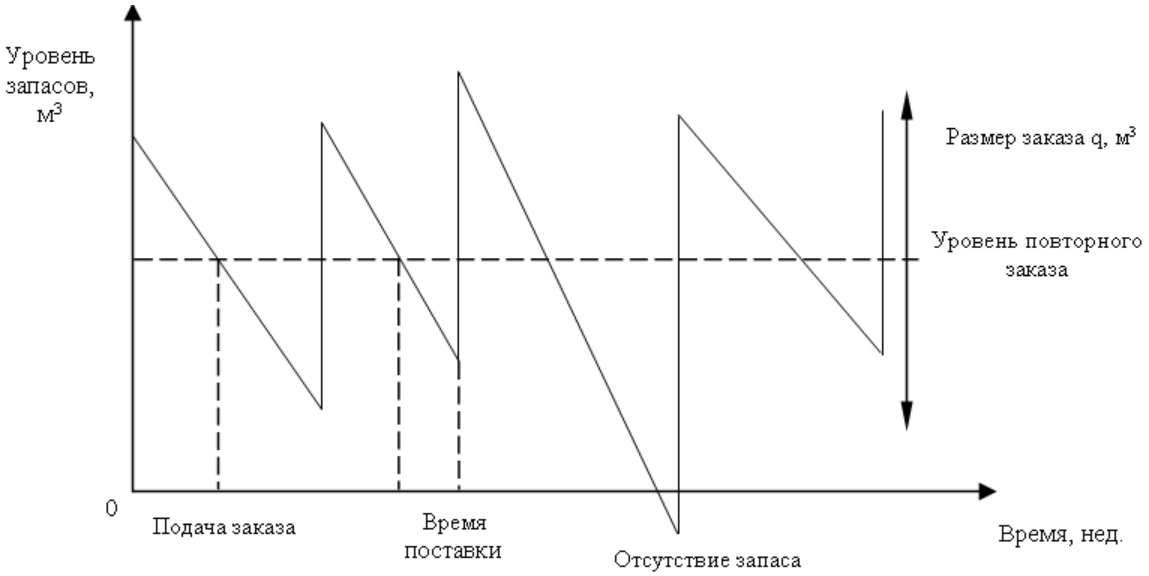


Рис. 1 - Стандартная модель хранения запасов

Для упрощения процесса моделирования в модель вводится ряд предпосылок:

1. Спрос на продукцию является постоянным, или приблизительно постоянным. Если коэффициент использования запасов является постоянным, то уровень запасов также будет уменьшаться с постоянным коэффициентом.
2. Предполагается, что время поставки известно и является постоянной величиной. Это означает, что заказ можно осуществлять в точке с определенными значениями временного параметра и размера запаса (уровень повторного заказа), которые обеспечивают получение заказа в тот момент, когда уровень запасов равен нулю.
3. Отсутствие запасов является недопустимым.
4. В течение каждого цикла запасов делается заказ на постоянное количество продукции (q).

Необходимо построить модель, которая описывает издержки, связанные с наличием запасов, за весь период их хранения. Длительность этого периода выберем равной одной

неделе. Для моделирования используем сглаживающую функцию. Введем следующую систему обозначений:

D - еженедельный спрос на запас продукции (м^3);

C_0 - переменная стоимость передачи на склад одного м^3, рублей/ м^3;

C_h - переменная стоимость хранения м^3 в запасе, рублей на м^3 в неделю;

q – объем заказа, м^3/заказ;

b – интенсивность потребления товара, м^3/нед;

t - рассматриваемый период времени;

tp - время поставки, дни.

TC - общая стоимость запаса м^3.

Рассмотрим подробно оптимальный размер заказа q_0 в неделю и интервал повторного заказа:

$TC(0)=q;$

$TC(t)=0.$

Экономичный размер заказа q_0 равен:

$$\frac{dTC}{dq}=0 \quad и \quad \frac{d^2TC}{dq^2}>0 \qquad (1)$$

$$\frac{dTC}{dq}=-C_0\frac{D}{q^2}+C_h\frac{1}{2} \qquad (2)$$

и

$$\frac{d^2TC}{dq^2}=-2C_0\frac{D}{q^3}+0>0, если \qquad q>0 \qquad (3)$$

Положим

$$\frac{DTC}{dq}=0, тогда \qquad -C_0\frac{D}{q^2}+C_h\frac{1}{2}=0 \qquad (4)$$

Следовательно

$$C_0\cdot\frac{D}{q^2}=C_h\cdot\frac{1}{2} \qquad (5)$$

$$q^2=\frac{2\cdot C_0\cdot D}{C_h} \qquad (6)$$

$$q_0=\sqrt{2\cdot C_0\cdot D/C_h} \qquad (7)$$

Формула (7) называется формулой Уилсона или формулой наиболее экономичного объема партии.

Интенсивность потребления запасаемого товара можно найти, разделив общее потребление товара на время, в течение которого он расходуется:

$$b=\frac{D\cdot tp}{t} \quad (м^3/нед) \qquad (8)$$

Поскольку величина спроса постоянна, количество товара, которое используется в течение поставки заказа, является одновременно и уровнем повторного заказа. Таким образом, новый заказ следует подавать, когда уровень запасов станет равным интенсивности потребления.[1]

Разрабатываемая информационная система позволят автоматизировать формирование заказов и учета товаров в региональных представительствах с учетом уровня запасов, повысить эффективность деятельности предприятия, производительности труда, улучшить качество обслуживания покупателей, сократить запасы товаров на складе.

Логическая структура базы данных определяется информационными потребностями проекта. При ее разработке выделяются основные информационные сущности предметной области, выявляются связи между ними. Затем логическая структура оптимизируется в соответствии с реализуемыми целевыми функциями проекта.

Разрабатываемая инфологическая модель данных приведена на рисунке 3.

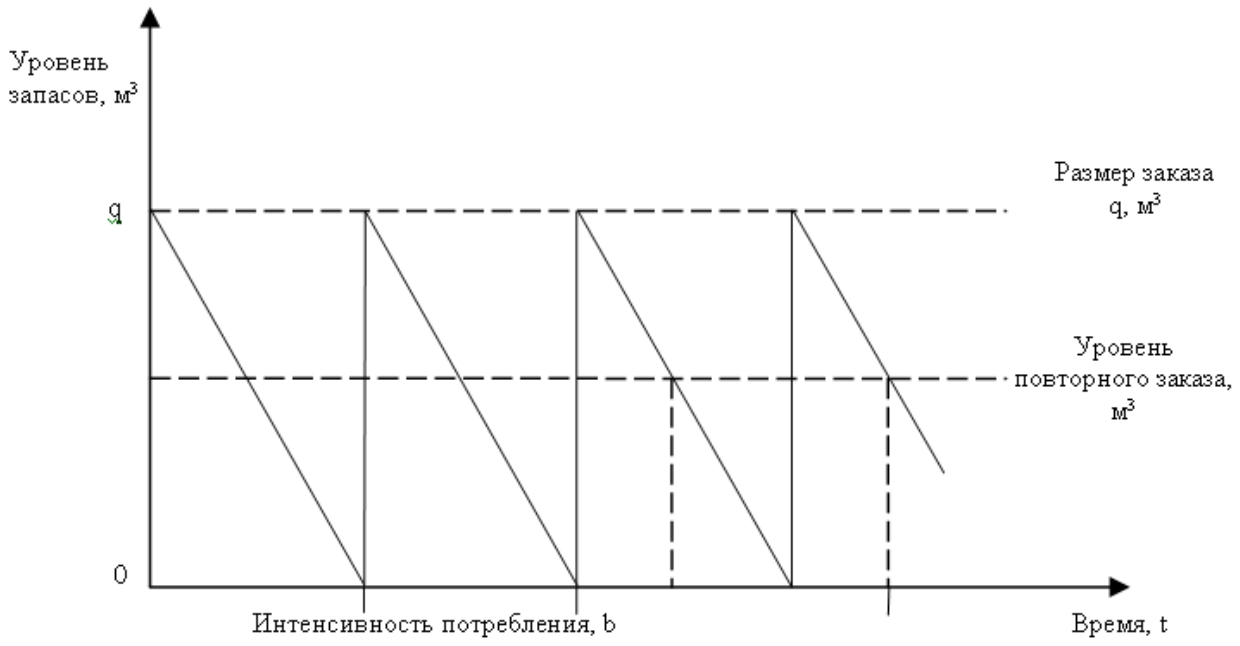


Рис. 2 - Уровень и интервал повторного заказа

Основными сущностями являются:

- Сущность «Сотрудник» характеризуется атрибутами – КодСотрудника, Фамилия, Пароль. Данная сущность связана с сущностями «Смены», «Приход» и «Продажа»;
- Сущность «Заявка» характеризуется атрибутами - Код заявки, № заявки. Данная сущность связана с сущностями «Сотрудник» и «Номеклатура»;
- Сущность «Смены» характеризуется атрибутами – КодСмены, КодСотрудника, Начало, Окончание. Данная сущность связана с сущностями «Сотрудник» и «Продажа»;
- Сущность «Приход товара» характеризуется атрибутами – КодПрихода, ДатаПрихода, КодНоменклатуры, Количество. Данная сущность связана с сущностями «Номенклатура» и «Сотрудник»;
- Сущность «Номенклатура» характеризуется атрибутами – КодНоменклатуры, Наименование, Цена, Товар. Данная сущность связана с сущностями «Приход» и «Продажа»;
- Сущность «Продажа» характеризуется атрибутами – КодПродажи, КодНоменклатуры, КодКонтрагента, Дата, Количество, Стоимость, КодСотрудника, КодСмены, Безнал. Данная сущность связана с сущностями «Номенклатура», «Сотрудник», «Смены» и «Контрагенты»;
- Сущность «Контрагенты» характеризуется атрибутами –КодКонтрагента, Наименование, Полное наименование, Безнал. Данная сущность связана с сущностью «Продажа».
- Сущность «Остатки» характеризуется атрибутами – КодОстатка, КодРП, КодНоменклатуры, Количество и Дата. Данная сущность связана с сущностями «Районы» и «Номенклатура»;
- Сущность «Районы» характеризуется атрибутами – КодРП, Название. Данная сущность связана с сущностями «Остатки» и «Обороты»;
- Сущность «Обороты» характеризуется атрибутами – Код, КодНоменклатуры, КодКонтрагента, Дата, Сумма, Количество, КодРП. Данная сущность связана с сущностями «Контрагенты» и «Районы»;
- Сущность «РасчетыКонтрагенты» характеризуется атрибутами – Код, КодКонтрагента, Сумма, Дата, КодРП. Данная сущность связана с сущностью «Контрагенты».

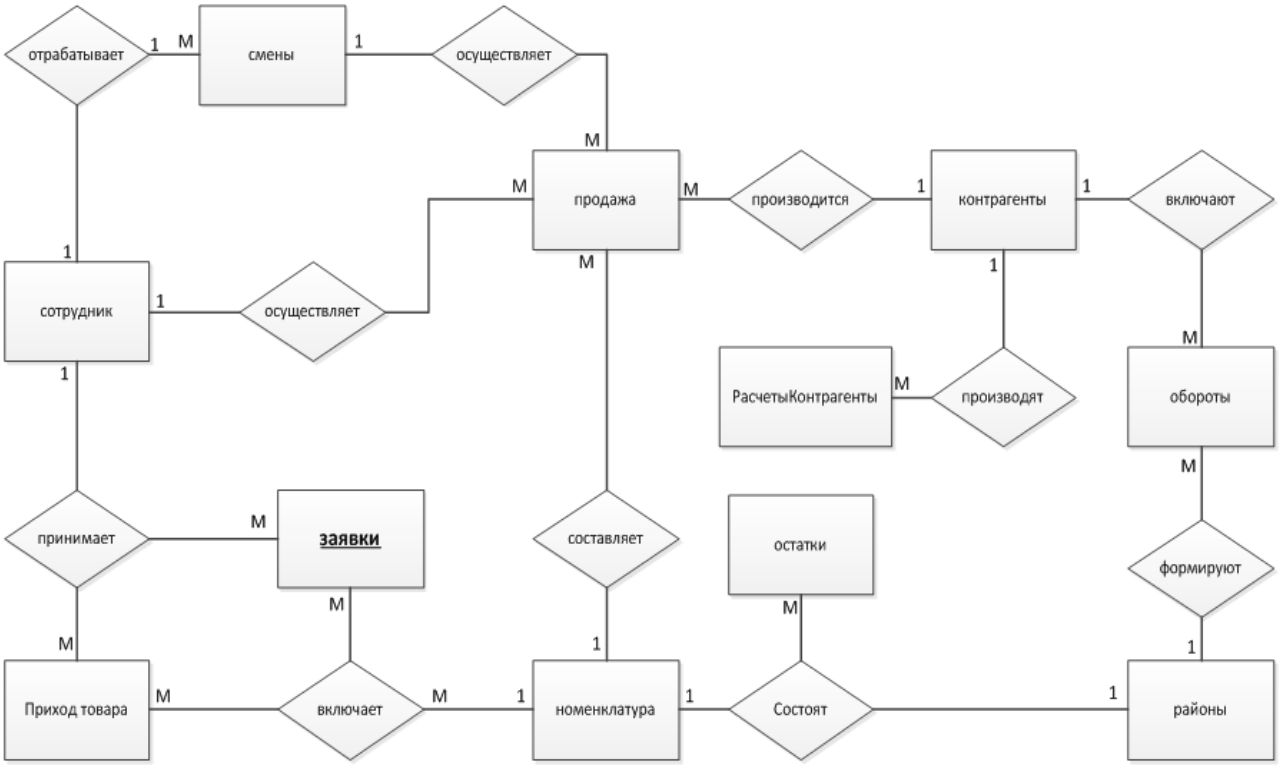


Рис. 3 - Инфологическая модель ЗАО «Аэробел»

Входной информацией для разрабатываемой информационной системы формирования заказов и учета товаров является:

Нормативно-справочная информация:

- Справочник Контрагенты. Список контрагентов, с которыми ЗАО «Аэробел» заключает договора на обслуживание по безналичному расчету.
- Справочник Номенклатура. Наименование товаров или услуг, и цены на них. Цена является единой по всем региональным представительствам.

Входная оперативная информация:

- Таблица Приход. Накладные на приход товара. В них указывается, кто принял товар, в каком количестве и дата. Цена товара заложена изначально в справочнике номенклатуры. Она не меняется, т.к. поставщик единственный. Количество запасов на складе.

Результатной информацией является:

- Обороты за смену: что продано, кому, дата и время продажи, на какую сумму и код заправки для идентификации в БД главного офиса.
- Сколько осталось товара в наличии на складе.

Автоматизация информационных потоков в региональных представительствах предприятия позволит упростить документооборот, создать единую информационную базу для анализа, эффективного управления запасами и контроля деятельности предприятия в данной сфере.

Литература

1. Исследование операций в экономике. Под ред. проф. Н.Ш. Кремера. – М.: Банки и биржи, ЮНИТИ, 1999. – 407 с.
2. Колесник А.П. Компьютерные системы в управлении финансами. - М.: «Финансы и статистика»,2002 г.
3. Кузнецов С.Д. СУБД (системы управления базами данных) и файловые системы.- М: Майор, 2001 г.

УДК 004.822, 004.891, 004.434:004.8

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ ПОСРЕДСТВОМ ИНТЕГРАЦИИ ТЕХНОЛОГИЙ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ И НОТАЦИЙ IDEF

Магомедова Мадина Нурмагомедовна, аспирант Санкт-Петербургского государственного экономического университета, ст. преподаватель кафедры “Информационных систем в экономике”, Санкт-Петербургский государственный экономический университет, филиал в г.Кизляре, Россия, Кизляр, dina0682@mail.ru

Проектирование информационных систем для бизнеса является сегодня наиболее распространенной сферой. Основной проблемой проектирования информационных систем является общение разработчика с заказчиками. Разработчик не всегда владеет всеми необходимыми знаниями предметной области, для которой он пишет приложение. Заказчик, далекий от программирования человек, просит реализовать алгоритм трудно формализуемых функций.

Сегодня уже есть предложения для решения таких проблем, такие как экстремальное программирование и проектирование, когда в процессе разработки задействованы обе стороны. Однако это не исключает возникновения разногласий между разработчиком и заказчиком. В рамках данного исследования предлагается решить проблему, доверив весь процесс проектирования приложения заказчику.

Проектирование информационных систем как сложный процесс охватывает различные стадии построения будущей системы. Для проектировщика особенно важно выявить объекты, которые входят в информационную систему предприятия. Объекты могут также включать абстрактные элементы системы.

Проектирование информационных систем будет состоять из следующих этапов:

- объектно-ориентированный анализ;
- объектно-ориентированное программирование;
- искусственный интеллект;
- нотации IDEF.

Объектно-ориентированный подход позволит объединить в себе алгоритм работы приложений и информационно-логическую модель предметной области. Как известно, «основополагающей идеей объектно-ориентированного подхода является объединение данных и действий, производимых над этими данными, в единое целое, которое называется объектом». [1]

Применение методологии объектно-ориентированного программирования подразумевает представление кода программы в виде символьной модели реальных объектов. В разрабатываемой концепции данный программный код будет генерироваться системой автоматически. Ввод процедур обработки не предусмотрен. Проектирование будет происходить только визуально, то есть отношение между модулями будущей информационной системы будут сформировываться автоматически.

Сегодня существуют концептуальные идеи по интеллектуализации разработки приложений для бизнеса - это объединение искусственного интеллекта и систем программирования.

Внедрение искусственного интеллекта в процесс проектирования подразумевает объединение нескольких компонентов в одну единую систему. Внедрение искусственного интеллекта для разработки прикладных программ подразумевает объединение методов объектно-ориентированного программирования и методологий построения бизнес-процессов в единую систему. Графический интерфейс такой среды проектирования информационных систем будет содержать инструменты построения потоков информации в организации. Интерфейс будет настолько прост, что пользователю не составит особого труда спроектировать основные компоненты, входящие в структуру его организации. На основе

сформированной структуры на интеллектуальном уровне системы автоматически генерируется программный код.

Программный уровень включает в себя программный код, генерируемый системой. Такая система будет отличаться от известных сегодня систем программирования, так как инструментарий системы разработки нового поколения будет намного интеллектуальней. Помимо проверки синтаксических ошибок программирования, такая система должна будет вести проверку приложения на семантическом уровне. То есть проверять правильность построения блоков приложения, организованных пользователем. [2]

Для проектировщика приложений особенно важно выявить объекты, которые входят в информационную систему предприятия. Как бы ни строился анализ, наиболее важным является практическая сторона построения приложений. Наиболее оптимальным сегодня средством отражения сущности информационных потоков является методология семейства IDEF для описания бизнес-процессов предприятий, на ее основе и будет строиться интерфейс взаимодействия пользователя с системой. Выбор данной методологии не является конечным. В качестве средства манипулирования объектами-составляющими информационной системы может быть выбран и другой язык, такой как UML. На данном этапе выбор методологии IDEF основывается на простоте его представления. При практической реализации концепции, описываемой в данном исследовании, может быть реализована новая методология, наиболее приемлемая в будущей интеллектуальной системе проектирования информационных систем.

Проектирование информационной системы подразумевает решение более широкого круга задач, в отличие от прикладных приложений. Так как же будет выглядеть интеллектуальная система проектирования информационных систем?

Информационная система - система обработки информации, работающая совместно с организационными ресурсами, такими как люди, технические средства и финансовые ресурсы, которые обеспечивают и распределяют информацию.[3]

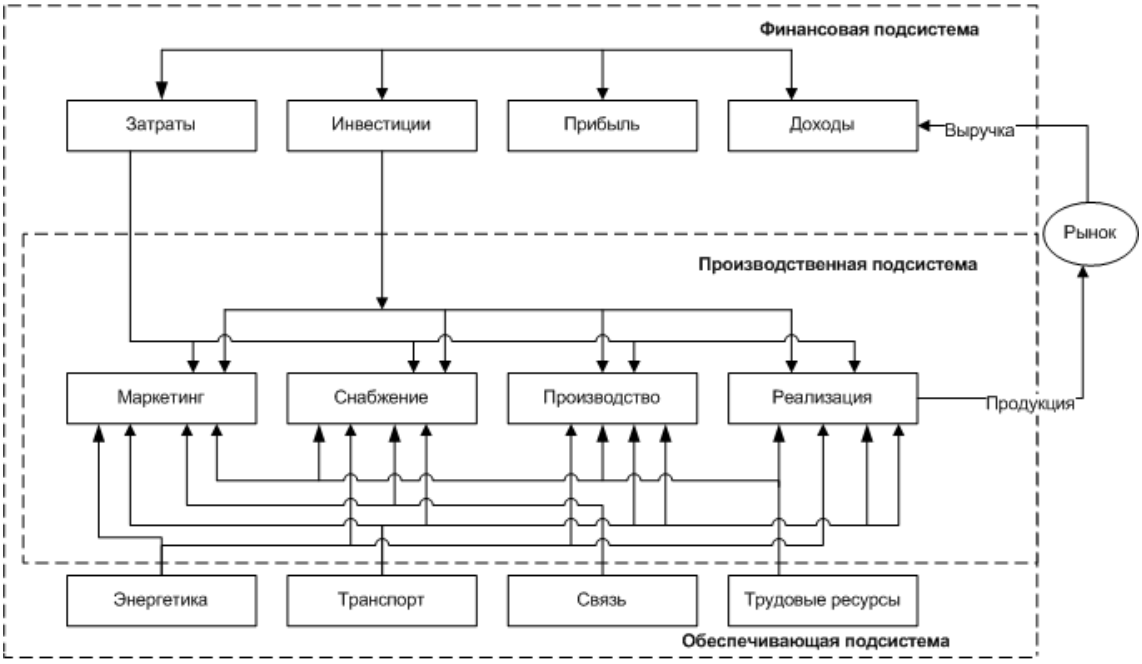


Рис. 1 – Система “Предприятие”

В качестве примера возьмем экономическую систему в рамках одного предприятия, предлагаемую автором Дрогобыцким И.Н. Любую экономическую систему можно рассматривать как образование, состоящее из трех подсистем или, как принято еще называть, систем второго порядка – производственной, финансовой и обеспечивающей.

Система «Предприятие» представлена на рисунке 1.[4]

Представим, что систему “Предприятие” необходимо будет реализовать с помощью предлагаемой в данной статье интеллектуальной среды проектирования. Проведя объектно-

ориентированный анализ, из предлагаемой системы можно выделить некоторые абстрактные и физические объекты, такие как:

- продукция;
- затраты;
- выручка;
- начисляемая заработная плата и т.д.

Все эти и другие объекты будут представлены для разработчика в проектируемой рабочей области. Процесс выбора будет схож с процессом построения бизнес-моделей с помощью нотации IDEF.

Реализация системы «Предприятие» с помощью предлагаемой нами интеллектуальной системы проектирования представлена на рисунке 2.

Рис. 2 – Взаимосвязь программных модулей

На данной схеме показано, что за каждым проектным модулем (инструментом) будет спрятан автоматически генерируемый программный код, из чего следует не обязательным требование знания основ программирования. Пользователь просто располагает на рабочей области необходимые для автоматизации модули.

Применение данной технологии позволит заметно интеллектуализировать процесс проектирования информационных систем, разрешит возможные разногласия между заказчиками и проектировщиками приложений.

Литература

1. Лафоре Р. Объектно-ориентированное программирование. Изд. Питер 2004г.
2. Магомедова М.Н. Объектно-ориентированный подход к проектированию прикладных приложений на базе искусственного интеллекта. Искусственный интеллект и его приложения: сборник материалов III Межвузовского научно-исследовательского семинара с международным участием, декабрь 2012 / Под ред. доц. Г.А. Лисьева, А.Л. Зленко. – Магнитогорск: МаГУ, 2012. – 163 с.
3. Стандарт ISO/IEC 2382-1
4. Дрогобыцкий Иван Николаевич. Системный анализ в экономике: учебник для студентов вузов, обучающихся по специальностям «Математические методы в экономике», «Прикладная информатика»/И.Н. Дрогобыцкий.-2-е изд., перераб. и доп. – М.: ЮНИТИ-ДАНА, 2011.-423с.

УДК 004.43:378.09

НАПИСАНИЕ СОБЫТИЙНО-ОРИЕНТИРОВАННЫХ ПРИЛОЖЕНИЙ ДЛЯ РАБОТЫ С БАЗОЙ ДАННЫХ

Мясникова Нелли Александровна, доцент, Южно-Российский государственный технический университет (Новочеркасский политехнический институт), Россия, Новочеркасск, mnela@list.ru
Сидоренко Андрей Андреевич, студент, Южно-Российский государственный технический университет (Новочеркасский политехнический институт), Россия, Новочеркасск, GhostProg666@yandex.ru

Для написания запросов к различным источникам данных программистам необходимо владеть соответствующим языком запросов. Язык запросов LINQ (Language Integrated Query) позволяет использовать единую модель для работы с различными форматами источников данных, упрощает написание запросов. Для написания запросов к базе данных была использована технология LINQ to SQL. Эта технология позволяет создать объектную модель, выраженную в языке программирования разработчика на основе реляционной модели базы данных. На рис. 1 показана получившаяся объектная модель (база данных содержит сведения об иностранных студентах ВУЗа).

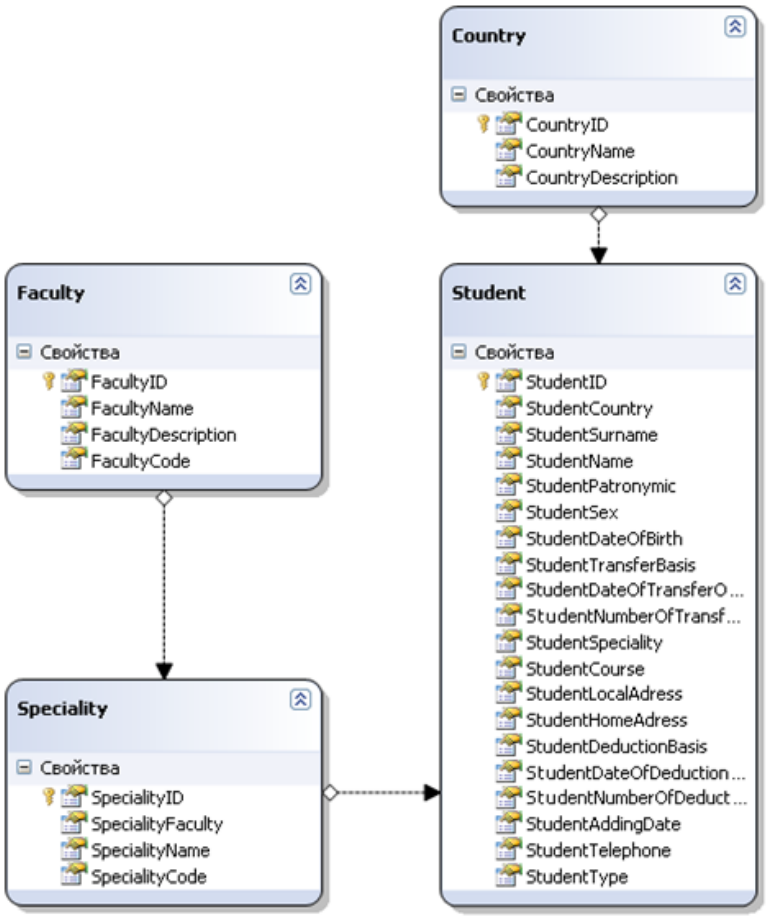


Рис 1. - UML – диаграмма предметной области

Для обращения к базе данных пишется запрос на языке LINQ, который далее транслируется в SQL и отправляется в базу данных. Затем база данных возвращает результат, который преобразуется в объекты, с которыми программист может работать на своем языке программирования.

Все запросы к базе данных были написаны в виде лямбда-выражений.

Лямбда выражения – это специальный синтаксис для объявления анонимных функций. Они особо полезны при написании запросов на языке LINQ.

Для написания лямбда выражения необходимо:

- Определить входные параметры и записать их слева от лямбда оператора =>
- Записать блок выражений справа от лямбда оператора =>

Рассмотрим простейший пример использования лямбда-выражения

```
delegate int Sum(int x, int y);
static void Main(string[] args)
{
  Sum A = (int x, int y) => x + y; // В данном примере лямбда-выражение
                                   // принимает два параметра
                                   // x и y и возвращает их сумму
  Console.WriteLine(A(2, 3));
```

```
    Console.ReadLine();
  }
```

На экран будет выведена сумма чисел 2 и 3. Пример написания запроса к базе данных

```
Speciality[] Temp1 = db.Speciality
                       .Where(m => m.SpecialityFaculty == FacultyID)
                       .ToArray();
```

Данное выражение следует понимать так:

"Выбрать из базы данных, из таблицы специальностей все специальности, у которых ID факультета равен ID выбранного факультета. Такая запись позволяет сократить количество кода, которое необходимо написать для создания запроса и обеспечивает наглядность"

Ниже представлено описание классов и структур, используемых в программе:

- Класс Student (хранит информацию о студенте)
  - StudentID – ID студента;
  - StudentCountry – страна, из которой приехал студент;
  - StudentSurname – фамилия студента;
  - StudentName – имя студента;
  - StudentPatronymic – отчество студента;
  - StudentDateOfBirth – дата рождения студента;
  - StudentTransferBasis – основание для зачисления;
  - StudentDateOfTransferOrder – дата приказа о зачислении;
  - StudentNumberOfTransferOrder – номер приказа о зачислении;
  - StudentSpeciality – специальность студента;
  - StudentCourse – курс на котором обучается студент;
  - StudentLocalAdress – местный адрес;
  - StudentHomeAdress – домашний адрес;
  - StudentDeductionBasis – основание для отчисления;
  - StudentDateOfDeductionOrder – дата приказа об отчислении;
  - StudentNumberOfDeductionOrder – номер приказа об отчислении;
  - StudentAddingDate – дата добавления студента в БД;
  - StudentTelephone – номер телефона студента;
  - StudentType – направление обучения.
- Класс Country (хранит информацию о стране)
  - CountryID – ID страны;
  - CountryName – название страны;
  - CountryDescription – описание страны;
  - Класс Faculty;
  - FacultyID – ID факультета;
  - FacultyName – название факультета;
  - FacultyDescription – описание факультета;
  - FacultyCode – шифр факультета;
- Класс Speciality
  - SpecialityID – ID специальности;
  - SpecialityName – название специальности;
  - SpecialityFaculty – факультет;
  - SpecialityCode – шифр специальности.

Для разработки приложения был выбран язык C# и платформа .NET. Этот выбор обоснован следующими четырьмя преимуществами:

1. Упрощен процесс установки и развертывания приложения. Компоненты .NET Framework не связаны с реестром. Поэтому установка подобных приложений сводится к копированию файлов в соответствующие каталоги и созданию ярлыков. Удаление приложений сводится к удалению файлов. Это позволяет пользователю корректно устанавливать и удалять приложения без необходимости вмешательства программиста.
2. При компиляции кода для .NET Framework генерируется не традиционный код (который состоит из процессорных команд), а код на промежуточном языке CIL. Во время выполнения приложения, CLR транслирует CIL в команды конкретного процессора. Это позволяет запускать приложение на любой машине, на которой установлена соответствующая версия .NET Framework.
3. Упрощенное повторное использование кода. Этот механизм позволяет создавать собственные классы, которые в дальнейшем можно без труда использовать при разработке других приложений
4. Автоматическое управление памятью позволяет в разы упростить управление использованием ресурсов ЭВМ и отыскивать во время выполнения программы объекты, которые больше не используются приложением и могут быть удалены.

В результате написания программы были получены навыки написания событийно-ореинтированных приложений для работы с БД, с использованием языка программирования C#. Для хранения данных была использована локальная база данных SQL Server Compact Edition. Обращение к базе данных были написаны на языке LINQ.

Литература

1. Окулов С.М. Основы программирования, Издательство: БИНОМ. Лаборатория знаний, 2010 г. ЭБС Книгофонд, http://www.knigafund.ru/books/127791
2. Нагинаев В.Н. Основы алгоритмизации и программирования на языке C++: Учебное пособие, Издательство: МИИТ, 2006 г. Книгофонд, http://www.knigafund.ru/books/8264
3. Г. С. Иванова. Объектно-ориентированное программирование. – МГТУ им. Н. Э. Баумана, 2002
4. Курс: Основы информатики и программирования. Лекция №10: Основы объектно-ориентированного программирования. Применение ООП к разработке программных проектов. Основные концепции ООП, www.intuit.ru/department/se/oip/10

УДК 004.451.8

ПЕРСПЕКТИВЫ РАЗВИТИЯ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ ОПЕРАЦИОННЫХ СИСТЕМ

Штанюк Антон Александрович, к.т.н., доцент кафедры информационных технологий в предпринимательской деятельности Нижегородского государственного университета им. Н.И. Лобачевского, shtan@land.ru

Введение

Использование объектно-ориентированного подхода к разработке операционных систем вызывало и вызывает дискуссии среди разработчиков [1]. Существуют области, в которых ООП широко применяется - например, графические интерфейсы пользователя, - и там у объектного подхода практически нет конкурентов. Вызвано это, на наш взгляд, огромным количеством кода, стоящим за отображением информации, а также событийным характером работы программ с GUI. Производительность интерфейсов не является главным требованием, в отличие, например, от дружественности пользователю. Операционная система работает в других условиях, и требования к производительности у неё могут перекрывать все остальные, что заставляет использовать «мультипарадигмальную смесь»

языков и подходов к разработке. Также немаловажную роль в этой области играют традиции, которые для GUI просто диктуют использовать ООП, а вот для ОС объектный подход всерьёз не рассматривается.

Объектно-ориентированные операционные системы (ОООС) в настоящее время – большая редкость, и зачастую представляют собой экспериментальную, лабораторную разработку. Говорить о создании действительно массовой ОООС общего назначения было бы неверно [2]. Но объектные черты у многих операционных систем всё же присутствуют.

1. Отличительные признаки ОООС

ОООС присущи следующие отличительные признаки:

1. Объектная архитектура. Основная сущность в системе – объект, и архитектура всей системы является совокупностью объектов, расположенных на разных уровнях в соответствии с иерархиями «содержать» и «являться».
2. Обмен информацией внутри системы. Обмен осуществляется в терминах посылки и приёма сообщений, которые осуществляют объекты.
3. Операции ввода/вывода при взаимодействии с устройствами также представляются в терминах объектов и посылки сообщений.
4. Минимум побочных эффектов. Объекты взаимодействуют друг с другом через сообщения, и их состояние тесно связано с поведением. В этом случае побочные эффекты запрещены.
5. Поддержка микроядерной концепции, позволяющей реализовать большинство функций ядра в виде сервисов, работающих в режиме задачи, а, следовательно, допускающих реализацию в виде автономных объектов, которыми проще управлять.
6. Распределённый характер, заключающийся в способности легко обмениваться программными объектами между различными компьютерами.
7. Основной язык разработки – C++, но возможны исключения в виде Java или даже Lisp.

Безусловно, между ОООС и обычными системами, широко распространенными в мире, существует много общего. Например, можно рассматривать unix-подобную ОС как объектную с точки зрения архитектуры, если под объектами подразумевать такие крупные компоненты как ядро, оболочки, утилиты. Да и в ядре существуют модули, драйверы, имеющие чёткий программный интерфейс взаимодействия с остальными частями системы. Создание новых процессов на основе клонирования существующих, описание взаимодействия процессов в терминах «отцы», «дети», «братья» - это имеет аналогию с концепцией наследования в традиционных ОО-программах. Всё это свидетельствует о стремлении разработчиков использовать преимущества объектного подхода к разработке ПО, но не диктует им стратегию разработки от начала до конца. В самом деле, в той же unix-системе границы между модулями ядра весьма условны, существует большое количество общих данных в пространстве ядра, а управляющие структуры процессов – это просто структуры данных, не имеющих специального объектного представления. В системе причудливо переплетаются процессы и потоки, файлы и каналы, обработчики прерываний и API-функции. Такую систему логичнее описывать как огромное здание, опутанное проводами и сетью труб, чем конгломерат взаимодействующих объектов.

2. Преимущества и недостатки ОООС

К достоинствам объектно-ориентированных ОС можно отнести:

- чёткость и прозрачность архитектуры, основанной на понятиях объекта и интерфейса;
- упрощение тестируемости в связи с разделением полномочий объектов и трассировкой сообщений;
- безопасность из-за возможности проводить более гибкую политику в отношении прав различных объектов;
- хорошая сопровождаемость благодаря более ясному исходному коду;

- расширяемость благодаря адаптивным интерфейсам на любом уровне системы.

К недостаткам ОООС можно отнести:

- более низкая производительность из-за наличия многих звеньев в цепи передачи сообщения от объекта к другому объекту, а также необходимости использования более абстрактных механизмов для поддержки ООП;
- существенный размер кода самой системы из-за необходимости описания интерфейсов объектов даже для самых простых действий.

3. Краткий обзор некоторых существующих ОООС

Сначала хотелось бы описать наиболее успешные версии ОООС.

BeOS и **Haiku** – две наиболее известные операционные системы, первая из которых появилась в середине 90-х годов, но из-за лицензионной политики не сумела «выжить» на рынке, а вторая представляет собой современную версию BeOS, написанную сообществом с нуля. Главной особенностью данных систем является активное использование языка C++ вместо C и объектно-ориентированный API. У Haiku есть большие шансы стать достаточно распространённой ОС для персональных компьютеров и её разработка в настоящее время идёт очень активно, хотя система пока только подошла к стадии вета-тестирования [3].

Genera – коммерческая операционная система, начало разработки которой относится к 1982 году. Разрабатывалась система компании Symbolic для LISP-компьютеров и писалась на языке программирования LISP. В качестве пользовательского интерфейса используется оригинальная разработка под названием Dynamic Windows, которая является полностью объектной. Современные версии системы работают поверх unix. Последняя версия появилась в 1998 году.

Athene – вышла в 2000 году. Разработчик Rocklyte Systems. Пользовательское окружение целиком состоит из объектов. Процессы могут размещать объекты в разделяемой памяти и использовать совместно. Разработка приложений под Athene подчиняется ООП-методологии. Проект закрылся в 2009 году.

Chorus – микроядерная система реального времени, которую разработала компания Sun для встраиваимых систем в 80-х годах прошлого века. Последние версии системы относятся к 1997 году и, судя по всему, исходный код системы был открыт компанией-разработчиком.

Choices – система, разработанная в университете штата Иллинойс, США. Написана на C++ и использует объекты для представления элементов ядра. Используется наследование для отделения платформонезависимых классов ядра от зависимых. Поддерживает архитектуры SPARC, x86 и ARM.

GEOS – операционная система, относящаяся к разряду легковесных с графическим интерфейсом пользователя. Создавалась как надстройка над DOS, подобно первым версиям Windows. По некоторым данным, написана на Object Assembler.

Mach – ядро операционной системы, которое разрабатывалось в качестве замены современным ядрам Unix в университете Carnegie-Mellon. Микроядерная архитектура с реализацией процесса в качестве объекта, обладающего набором системных ресурсов, а также обмен сообщений между объектами ядра, позволяет отнести Mach к важным элементам ОООС. Данный проект оказал большое влияние на архитектуру ядер многих современных ОС, среди которых GNU Hurd, NEXTSTEP, MAC OS X и ряд других.

Peace – распределённая ОС, предназначенная для работы на вычислительных кластерах с большим количеством процессоров и ядер и наиболее подходящая в области параллельных вычислений.

Spring – операционная система, разработанная компанией Sun в начале 90-х годов. Относится к числу микроядерных и оказала некоторое влияние на язык Java и ОС Solaris.

TAJ – операционная система, разработанная на C++ в Индии и работающая на архитектуре x86. Поддерживает многозадачность, многопоточность, многопользовательский режим.

Существуют также несколько систем, написанных на языке Java и выполняющихся на виртуальной машине (jvm). К ним относятся *JavaOS, JOS, JNode* и *JX*.

JavaOS – операционная система компании Sun, разработанная в 1996 году. Код системы написан преимущественно на языке Java и может выполняться на множестве аппаратных платформ. В системе есть графический интерфейс, основанный на библиотеке AWT.

JNode – современная операционная система, написанная на языке Java в 1995 году и развивающаяся до настоящего времени. Обладает микроядром, написанным на ассемблере, и работающей поверх него JVM.

Следует отметить тот факт, что практически все представленные в данном списке системы (кроме BeOS и Haiku) никогда не были известны широким массам пользователей и даже за стенами исследовательских лабораторий, где были разработаны. Разработка многих систем была прекращена из-за финансовых проблем или в связи с переключением на более актуальные задачи.

4. Основные концепции ООП при разработке архитектуры ОООС

В результате следования принципу инкапсуляции все системные объекты скрывают в себе ресурсы и основные абстракции для работы с ними (файл, процесс, память…), сокращая при этом количество связей за счёт отделения интерфейса от реализации.

Основным способом обмена информации в системе является передача сообщений от одних объектов к другим. Благодаря расположению объектов на различных иерархических уровнях, удаётся наиболее чётко прописать их взаимодействие при решении определённых задач.

В результате использования наследования, системные объекты используют одинаковые наборы методов соответствующих классов. Классы-наследники имеют возможность переопределять родительские методы или расширять их состав. Системные объекты, вызывая методы друг друга, передают скрытые указатели, позволяющие установить принадлежность метода. Это даёт возможность контролировать права объектов на уровне передачи сообщений. Специальный системный объект ObjectProxy управляет ссылками на системные объекты и транслирует сообщения в системе, работая при этом в привилегированном режиме процессора [4].

5. Перспективы развития

На основе анализа существующих ОООС можно наметить несколько путей их развития.

Первый путь: существование в виде полностью независимых ОС для аппаратной платформы (или платформ). Развитие по данному пути осложнено необходимостью создавать аппаратно-зависимые модули, написанные на языке ассемблера и поддерживать большой спектр современного оборудования, что является трудоёмким делом. Вряд ли такая система в обозримом будущем будет успешно конкурировать с распространёнными на рынке гибридными ОС и лучше приспособленными к аппаратным новинкам. Но можно предполагать, что в дальнейшем ОООС всё-таки получат широкое распространение благодаря своим конструктивным преимуществам.

Второй путь: существовать в виде объектно-ориентированных надстроек к имеющимся ядрам или ОС. Различные системы на основе Java иллюстрируют данное направление.

Третий путь: трансформация существующих ОС или их частей, постепенное внедрение объектного подхода в существующие решения. Одним из ярких представителей этой группы систем является Haiku, в которой реализован объектный API-интерфейс, вызывающий активный интерес [3].

Литература

1. Фрактальная ОС [Электронный ресурс]: http://tunilab.org/UniEnv/FractalOS

2. Википедия [Электронный ресурс]: http://en.wikipedia.org/wiki/Object-oriented_operating_system
3. Рассвет Haiku [Электронный ресурс]: http://habrahabr.ru/post/143176
4. Vivek Kumar Singh "Object oriented operating system"[Электронный ресурс]: http://bhu.ac.in/ComputerScience/vivek/OO-OS.ppt

УДК 004.94

ПРОФИЛЬ UML ДЛЯ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ

Гурьянов Василий Иванович, к.т.н, доцент, Филиал Санкт-Петербургского государственного экономического университета в г.Чебоксары, Россия, Чебоксары, vg2007sns@rambler.ru

Введение

В последние годы в сфере имитационного моделирования отмечается возрождение интереса к имитационному моделированию на универсальных языках программирования. Естественным представляется применение таких современных методологий разработки ПО, как, например, UP [1]. Таким образом возникает задача разработки профиля UML, который разрешит использование UML-средств для разработки имитационных моделей. Однако на этом пути появляются определенные методологические трудности.

Дело в том, что в сфере имитационного моделирования имеет место уникальная ситуация. С одной стороны, UML можно использовать как язык описания имитационной модели (подобно RUP Business Modeling), с другой – как язык описания программных систем. Избежать этой двойственности невозможно. Возникает проблема непротиворечивости обеих моделей, порождая тем самым проблему *встраиваемости* методологии разработки имитационных моделей в методологию UP. В работе [2] предложен проект профиля (далее - *Scientific Profile* или *UML SP*), который, на наш взгляд, решает эту проблему. Профиль был проверен на ряде задач имитационного моделирования (см. [2-11]). На данный момент существует настоятельная необходимость в формальном определении этого UML-языка. В данной работе приведена метамодель *UML SP* и приведена аргументация в пользу предлагаемого решения. Приведен также краткий обзор опыта построения объектных имитационных моделей на этом языке.

1. Концептуальная модель

Мы будем в основном опираться на версию UML 1.5, т.к. выразительных средств этой линейки UML для целей имитационного моделирования, по меньшей мере, на данный момент оказалось достаточно. Что касается UML 2.0, то профиль подобный *UML SP* может потребоваться для разработки сложных имитационных моделей, использующих современные технологии, такие как MDA.

Концептуальная модель *Scientific Profile* выражает две основные концепции. Предполагается, что описание объекта имитационного моделирования представлено в форме глоссария. Профиль предоставляет регулярный механизм назначения предметной семантики программным сущностям. Для этого используется механизм помеченных значений для стереотипов в виде {Concept = имя концепта}. Тем самым вводится двойственная семантика. Чтобы разграничить семантики, будем говорить о *вычислительной семантике* и о *предметной семантике*. Концепты верхнего уровня фиксируются в названиях стереотипов и определяют наиболее общие понятия, которых достаточно для описания любой ситуации исследования имитационной модели.

2. Применение UML SP

Профиль построен на метаклассах UML и может использоваться в различных методологиях ООАП, однако мы будем полагать, что используется UP.

Профиль предназначен для разработки имитационных моделей и поэтому рассматриваются только те требования к ПО, которые отражают процесс исследования имитационной модели. В частности, это находит отражение в модели прецедентов. По завершению разработки имитационной модели, модели *UML SP* включаются (точнее – интегрируются) в аналогичные модели UP (см. рис.2). Это обеспечивает согласованность UP и процесса разработки имитационной модели.

Вернемся к проблеме, очерченной в начале статьи. В случае имитационного моделирования необходимо четко различать предметную область и объект имитационного моделирования. Модель предметной области определяет существенные объекты контекста функционирования ПО имитационного моделирования (ПОИМ). Модель объекта изучения вообще не входит в модель предметной области и является самостоятельной моделью. Конечно, если ПОИМ используется для мониторинга или управления неким объектом, то этот объект как-то будет представлен в модели предметной области. Но это уже другой случай.

С тем, чтобы обеспечить встраиваемость и согласованность имитационной модели с UP, в профиле модель анализа предполагает двойственную семантику. Сущности, входящие в эту модель участвуют в двух разных процессах: (а) в рабочем потоке анализа UP и (б) в процессе создания имитационной модели. Это схожие процессы (как по форме, так и по методам), однако область их приложения совершенно разная: первый – работает с моделью предметной области, второй – с моделью объекта имитационного моделирования. По этой причине имена не могут использоваться так же, как это делается в схожих по назначению UML-профилях, таких как RUP Business Modeling. Имена программных сущностей, входящих в модель анализа, образуют словарь анализа и для обеспечения непротиворечивости имеют *вычислительную семантику*. Концепты образуют самостоятельный словарь и имеют *предметную семантику*. Модель прецедентов и модель дизайна оперируют с сущностями, имеющими *вычислительную семантику*, и поэтому им не приписываются концепты. Однако обе модели входят в процесс создания имитационной модели. В тоже время эти модели – часть подобных моделей UP.

3. Метамодель профиля

UML SP использует все элементы UML. Профиль содержит определения стереотипов (см. пример на рис.1). Все термины *UML SP* выделены наклонным шрифтом. Важным аспектом применения профиля является возможность создания субпрофилей, конкретизирующих стереотипы; это также отображено на рис.1.

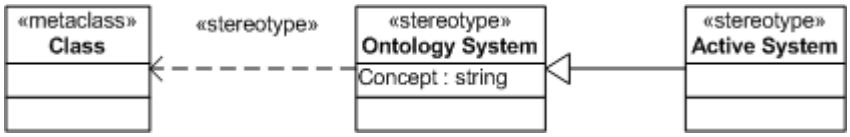


Рис.1 - Пример определения стереотипов Ontology System и Active System

Epistemology Entity "metaClass" Package (из Model Management)

Предметная семантика. Методологии проведения измерений, выраженные в процедурах измерений и обработки данных. В частности, включает методы статистической обработки данных. *Вычислительная семантика.* Иерархия абстрактных классов, пригодных для повторного применения. *Помеченные значения.* Категория: Attribute; Имя: Concept; Тип: String; Документация: Название системы мер.

Ограничения. Пакет используется пакетом «*Research Instruments*»

Exist "metaClass" Operation (из Core)

Предметная семантика. Стереотип помечает те операции класса, которые определяют единицы дискретно-событийного времени имитационной модели. Этот стереотип выражает принцип, который может быть назван *принципом объектно-темпоральной декомпозиции*: на каждом шаге объектной декомпозиции необходимо определять деятельность, которая задает единицу дискретно-событийного времени для каждого из выделенных объектов.

Вычислительная семантика. Метод класса. *Помеченные значения*. Нет. *Ограничения.* Применим к операциям классов, входящих в пакеты *World* и *Ontology Entity*.

Measurement "metaClass" Dependency (из Core)

Предметная семантика. Взаимодействие исследовательской установки и объекта исследования. *Вычислительная семантика*. Контролируемое нарушение инкапсуляции классов пакета «*World*» *Помеченные значения*. Категория: Attribute; Имя: Concept; Тип: String; Документация: Название эксперимента или наблюдения.

Ограничения. Стереотип *Measurement* применяется к отношениям зависимости, связывающие пакеты *Research Instruments* и *World*. Пакет *Research Instruments* зависит от *World*.

Make "metaClass" Operation (из Core)

Предметная семантика. Прагматика сообщения. *Вычислительная семантика.* Метод класса. Стереотип, применяемый ко всем операциям классов по умолчанию. Применим ко всем операциям, которые не помечены стереотипом *Exist*.

Помеченные значения. Категория: Attribute; Имя: Concept; Тип: String; Документация: Имя прагматики сообщения. *Ограничения.* Применим к операциям классов, входящих в пакеты «*World*» и «*Ontology Entity*».

Ontology Activity "metaClass" State (из Core, Activity – состояние-действие)

Предметная семантика. Описывает ситуацию, когда в каком-либо контексте некоторый процесс занимает пренебрежимо малое время и может рассматриваться как событие. С точки зрения наблюдателя контекста, процесс представляет собой событие, не имеющее продолжительности. С точки зрения наблюдателя процесса, контекст представляет собой статическую картину, а изменение доступных переменных контекста запрещено.

Вычислительная семантика. В языках программирования активность моделируется процедурой; создание активности – это вызов процедуры из текущей процедуры

Помеченные значения. Категория: Attribute; Имя: Concept; Тип: String; Документация: Имя процесса. *Ограничения.* Только в пакетах «*World*» и «*Ontology Entity*».

Ontology Atom "metaClass" Class (из Core).

Предметная семантика. Конечный уровень декомпозиции изучаемой системы на подсистемы. *Вычислительная семантика.* Программные классы.

Помеченные значения. Категория: Attribute; Имя: Concept; Тип: String; Документация: Наименование атомарного объекта. *Ограничения.* Программным сущностям, определяющим строение классов с этим стереотипом, нельзя назначать концепты.

Ontology Environment "metaClass" Class (из Core)

Предметная семантика. Определяет функцию изучаемой системы относительно окружающей среды. В простых случаях достаточно задать граничные и начальные условия.

Вычислительная семантика. Программный класс.

Помеченные значения. Категория: Attribute; Имя: Concept; Тип: String; Документация: Наименование контекста исследуемой системы.

Ограничения. Класс «*Ontology Environment*» находится в отношении агрегации с классом «*Ontology System*».

Ontology Entity "metaClass" Package (из Model Management)

Предметная семантика. Классификационная система, в которую входит объект моделирования. *Вычислительная семантика.* Иерархия абстрактных классов, пригодных для повторного использования в линейке имитационных моделей. Пакет, как правило, имеет сложную структуру.

Помеченные значения. Категория: Attribute; Имя: Concept; Тип: String; Документация: Наименование классификационной системы, в которую входит объект исследования.

Ограничения. Пакет используется пакетом «*World*». Должен содержать, по меньшей мере, два класса «*Substance*» и «*Ontology Space*».

Ontology Space "metaClass" Class (из Core).

Предметная семантика. Определяет мерономию классификационной системы и описывает мероны. Позволяет моделировать пространство состояний системы.

Вычислительная семантика. Иерархия классов, задающая пользовательские типы.

Помеченные значения. Категория: Attribute; Имя: Concept; Тип: String; Документация: Наименование мерона классификационной системы. *Ограничения.* Только для внутренних переменных классов «*Substance*» и классов пакета «*World*».

Ontology System "metaClass" Class (из Core).

Предметная семантика. Изучаемая система. *Вычислительная семантика.* Программный класс. Классы, помеченные стереотипами *Ontology Environment, Ontology System* и *Ontology Atom*, образуют тройку объектной декомпозиции и находятся в отношении «часть-целое». Декомпозиция на подсистемы может применяться рекурсивно.

Помеченные значения. Категория: Attribute; Имя: Concept; Тип: String; Документация: Наименование изучаемой системы. *Ограничения.* Класс «*Ontology System*» находится в отношении агрегации с классом «*Ontology Atom*».

Research Analysis Model "metaClass" *Model* (из Model Management)

Предметная семантика. Имитационная модель в широком смысле.

Вычислительная семантика. Описывает реализацию *прецедентов*, моделируя взаимодействие между компонентами имитационной модели.

Помеченные значения. Категория: Attribute; Имя: Concept; Тип: String; Документация: Название имитационной модели.

Ограничения. Не может содержать элементы, входящие в *Research Design Model*.

Research Design Model "metaClass" *Model* (из Model Management) и ***Research Design Model Realization*** "metaClass" *Collaboration* (из *Collaboration*).

Семантика. Модель анализа не зависит от конкретного языка программирования. Модель дизайна определяет способ описания модели анализа для выбранного языка реализации. Композиции параллельных процессов модели анализа представлена в модели дизайна как квазипараллельный процесс. В работе [9] высказано предположение, что достаточно потребовать эквивалентности процессов в форме наблюдаемой конгруэнции по Р. Милнеру. *Ограничения.* «*Research Design Model»* требует обязательной реализации взаимосвязи с «*Research Analysis Model*».

Research Instruments "metaClass" Package (из Model Management)

Предметная семантика. Описывает программные сущности, моделирующие средства наблюдения и измерений. *Вычислительная семантика.* Пакет для конкретных классов.

Помеченные значения. Категория**:** Attribute; Имя**:** Concept; Тип**:** String; Документация**:** Название измерительной системы, системы мониторинга или экспериментальной установки.

Ограничения. Не может содержать элементы, входящие в пакеты «*World*» *или* «*Epistemology Entity*».

Researcher "metaClass" Actor (из Use Case)

Семантика. Кто-то или что-то вне имитационной модели, взаимодействующий с имитационной моделью с целью исследования имитационной модели. Среди ролей *Researcher* наиболее важными являются роли *Наблюдатель контекста* и *Наблюдатель системы*. *Ограничения.* Может быть ассоциирован только с «*Research Use Case*»

Research Use Case "metaClass" *Use Case* (из Use Case)

Семантика. Каждый экземпляр прецедента описывает последовательность действий, связанных с изучением моделируемой системы.

Ограничения. Только «*Researcher*» может взаимодействовать с «*Research Use Case*»

Research Use Case Model "metaClass" Model (из Model Management)

Семантика. *Research Use Case Model* - это модель предполагаемых функций имитационной модели. Используется для определения части функциональных требований. На рис.2 показано типичное соотношение прецедентов имитационной модели и ПОИМ.

Помеченные значения. Нет

Ограничения. Модель должна содержать «*Researcher*» и «*Research Use Case*».

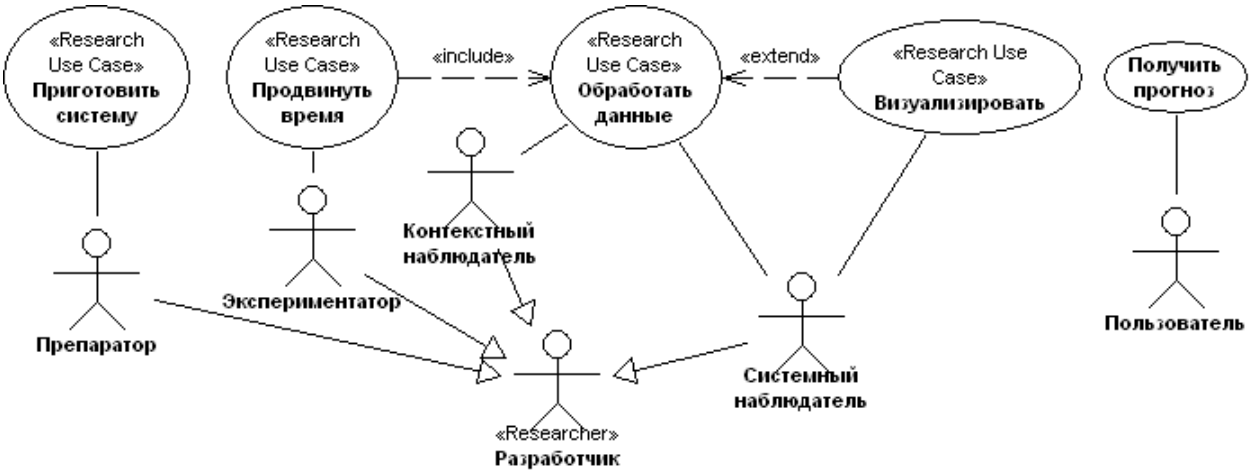


Рис.2 - Интеграция диаграммы *Research Use Case* в UP

Research Use Case Realization "metaClass" *Collaboration* (из *Collaboration*)

Предметная семантика. Схема экспериментальных действий.

Вычислительная семантика. Определяет реализацию прецедентов «*Research Use Case*» с точки зрения программной системы.

Ограничения. Требует обязательной реализации взаимосвязи с «*Research Use Case*».

Substance "metaClass" Class (из Core).

Предметная семантика. Классы, моделирующие таксоны. *Вычислительная семантика.* Это может быть единственный класс, задающий общий интерфейс для конкретных классов.

Помеченные значения. Категория: Attribute; Имя: Concept; Тип: String; Документация: Наименование таксонов классификационной системы. *Ограничения.* Нет.

Universe "metaClass" Package (из Model Management)

Предметная семантика. Все множество сущностей, связанных с данным исследованием. *Вычислительная семантика*. Представление «*Research Analysis Model*».

Помеченные значения. Категория**:** Attribute; Имя**:** Concept; Тип**:** String; Название имитационной модели. *Ограничения.* Должен содержать элементы UML SP.

World "metaClass" Package (из Model Management)

Предметная семантика. Стереотип *World* (мир модели или *модельный мир*) отражает исследуемую систему и ее окружение и содержит описание имитационной модели в традиционном понимании (имитационная модель в узком смысле слова).

Вычислительная семантика. Конкретные классы данной имитационной модели.

Помеченные значения. Категория: Attribute; Имя: Concept; Тип: String; Документация: Название модельного мира. *Ограничения.* Пакет должен содержать «*Ontology Environment*», «*Ontology System*», «*Ontology Atom*».

4. Примеры объектных имитационных моделей на UML SP

С примерами применения *UML SP* можно ознакомиться в работах [2-11]. В работе [3] предложена объектная модель региона, основу которой составляет макроэкономическая модель Дж. М. Кейнса. В работе [6] дано краткое описание субпрофиля для теории активных систем. Данный субпрофиль предназначен для имитационного моделирования бизнес-процессов и организационных систем. В работе [8] предложена объектная имитационная модель хронотопа, которая позволяет моделировать сложное поведение систем. Программная конструкция использует паттерн *State*. В работе [7] рассмотрена объектная имитационная модель социальных конфликтов. Рассматривались также модели бизнес-процессов и модели из области биологии, физики, социальной психологии, маркетинга. В работах [10, 11] предложена модель самовоспроизведения фирмы для процессов франчайзинга. Архитектурный паттерн - *Visiter*. В работе [4] рассматривается схожая

проблема – моделирование самовоспроизведения систем на основе механизма самоорганизации.

В работе [5] с позиций конструктивной математики рассматриваются объектные имитационные модели классических фракталов. Предложена архитектура, таксономия пакета «*Ontology Entity*», диаграмма классов. Предложена конструктивная процедура. Предложена измерительная процедура для вычисления размерности Хаусдорфа.

Для разработки объектных имитационных моделей использовались системы программирования Borland C++ Builder и Cincom Smalltalk (VisualWorks 7.6).

Выводы

В данной статье ставилась задача разработки профиля UML, которая позволит использовать методологию UP для разработки имитационных моделей. В работе представлена метамодель профиля. Опыт разработки конкретных объектных имитационных моделей показал, что достаточно тех стереотипов, которые приведены в данной версии профиля. Однако нельзя исключать существование контрпримеров, что потребует внесения изменений в существующую метамодель.

Литература

1. Арлоу Д., Нейштадт И. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2007. – 624 с.
2. Гурьянов В.И. Специальный UML-профиль для моделирования сложных систем // Информационные технологии моделирования и управления. – Воронеж: Изд-во «Научная книга», 2010. – № 3(62). – С. 356-362.
3. Гурьянов В.И., Шишкин В.С. Объектное моделирование социально-экономических систем //Современные проблемы прикладной информатики: Сб.науч.трудов Международной научно-практической конференции по современным проблемам прикладной информатики. 25-27 мая 2010 г / отв.ред.И.А.Брусакова, Е.Н.Панова – СПб: Изд-во Политехнического университета, 2010. –С. 160-165
4. Гурьянов В.И. Обобщение грамматик Линденмайера на организмы с иерархической структурой // АВСИ-2011, Том 2. – Коломна: МГОСГИ, 2011. – С.20-23. (см. http://www.informatika.mgosgi.ru/files/conf2011/5/Guryanov.pdf)
5. Гурьянов В.И. Объектное моделирование фрактальных структур // Математические модели и их приложения: сб. науч. тр. Вып. 13. – Чебоксары: Изд-во Чуваш. ун-та, 2011.- С. 148-159
6. Гурьянов В.И. Логический подход в методологии имитационного моделирования активных систем // Имитационное моделирование. Теория и практика: Сборник докладов пятой юбилейной всероссийской научно-практической конференции ИММОД-2011. Том 1. СПб.: ОАО «ЦТСС». 2011. – с.129-133. (см. http://www.simulation.su/files/immod2011/material/18.pdf)
7. Гурьянов В.И. Имитационное моделирование конфликтов в общественных и экономических системах // Математические модели и их приложения: сб. науч. тр. Вып. 14. – Чебоксары: Изд-во Чуваш. ун-та, 2012.- С. 136-141
8. Гурьянов В.И. Моделирование сложного поведения общественных и экономических систем // Всероссийская научно-практическая конференция по имитационному моделированию социально-экономических систем (ВКИМСЭС). ГОУ ВПО ВЗФЭИ. Труды конференции 15 мая 2012г., Москва: ООО «Принт-Сервис», 2012. – С. 125-129. (http://simulation.su/uploads/files/default/vkimses-2012-125-129.pdf)
9. Гурьянов В.И. Обратная задача распараллеливания в имитационном моделировании // АВСИ-2012.. – Коломна: МГОСГИ, 2012. – С.187-191. (см. http://www.informatika.mgosgi.ru/files/conf2012/5-6/Guryanov.pdf)
10. Гурьянов В.И. Объектная модель процессов самовоспроизведения фирмы // Интеллектуальные системы принятия решений и проблемы вычислительного интеллекта: материалы международной научной конференции (ISDMCI-2012). – Херсон: ХНТУ, - 2012 - С. 65-67.
11. Гурьянов В.И. Изменчивость в объектной модели самовоспроизведения фирмы. // Математическое и имитационное моделирование систем (МОДС 2012): материалы седьмой

международной научно-практической конференции, - Чернигов-Жукин: ЧГТУ - 2012 - С.87-91. (см. http://simulation.su/uploads/files/default/2012-conf-mods-87-90.pdf)

УДК 004.891

РАЗРАБОТКА ИНТЕЛЛЕКТУАЛЬНОЙ СИСТЕМЫ ЭНЕРГОМЕНЕДЖМЕНТА НА ОСНОВЕ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПОДХОДА

Иванов Сергей Александрович, студент, Комсомольский-на-Амуре государственный технический университет, Россия, Комсомольск-на-Амуре, kotolapsys@gmail.ru
Вяль Леонид Андреевич, студент, Комсомольский-на-Амуре государственный технический университет, Россия, Комсомольск-на-Амуре, leoneedos91@mail.ru
Горькавый Михаил Александрович, канд. техн. наук, доцент, Комсомольский-на-Амуре государственный технический университет, Россия, Комсомольск-на-Амуре, idpo@knastu.ru

Введение

В условиях активного развития промышленности и бурного роста численности населения планеты возникла проблема истощения исчерпаемых источников природных ресурсов. Вследствие этого наблюдается тенденция к росту уровня потребления энергоресурсов. Разработка и внедрение системы энергоменеджмента (СЭМ) является первоочередным энергосберегающим мероприятием для множества предприятий, так как только за счет беззатратных (организационных) и малозатратных мероприятий позволяет сократить энергопотребление до 30% (по данным Минэнерго). [2].

В настоящее время в Комсомольском-на-Амуре государственном техническом университете ведутся работы по разработке и внедрению объектно-ориентированной СЭМ обеспечивающей оперативность принятия управленческих решений, направленных на снижение потребления топливно-энергетических ресурсов (ТЭР).

Цель создания СЭМ - снижение на предприятии удельных показателей потребления ТЭР и оптимизация использования финансовых ресурсов для реализации энергосберегающих проектов.

Для эффективного функционирования СЭМ на предприятии должна учитывать всех агентов и связи между ними, поэтому СЭМ является сложной системой, как в реализации, так и в функционировании. Для разработки механизмов построения СЭМ был выбран объектно-ориентированный подход, т.к. он позволяет применять мощные выразительные средства объектного и объектно-ориентированного программирования, использующего в качестве блоков классы и объекты, также в объектной модели отражается и множество других факторов. Объектный подход является унифицирующей идей всей компьютерной науки, применимой не только в программировании, но также в проектировании интерфейса пользователя, баз данных и даже архитектуры компьютеров. Поэтому применение объектно-ориентированного подхода к проектированию системы значительно упрощает ее программную реализацию. Ориентация на объекты позволяет справляться со сложностью систем самой разной природы. В объектно-ориентированном анализе классификация или определение общих свойств объектов, помогает найти общие ключевые абстракции и механизмы, что, в свою очередь, приводит к более простой архитектуре системы.

Разработка СЭМ осуществляется на основе и с учетом имеющихся на предприятии организационных, технических и информационных ресурсов: автоматизированных систем управления технологического процесса (АСУ ТП), автоматизированной системы контроля и учёта энергоресурсов (АСКУЭ), метрологических средств, средств ВТ, средств телекоммуникации, структуры отделов и служб, связанных с вопросами энергоэффективности, действующих положений, в том числе положения о материальном стимулировании за эффективное использование ТЭР, накопленных массивов информации об энергопотреблении. Данное решение оптимально для реализации в учебном заведении, за

счет минимальных затрат на реализацию и видимого эффекта энергосберегающих мероприятий.

Именно данный подход целесообразен для ВУЗа, так как позволит добиться высоких показателей энергосбережения при минимальных затратах. Финансовая сторона энергосберегающих мероприятий крайне важна для учебного заведения, так как его бюджет строго регламентирован.

1. Описание концептуальной модели СЭМ

Была разработана концептуальная модель СЭМ, которая раскрыта на рисунке 1. На схеме указанны только основные связи.

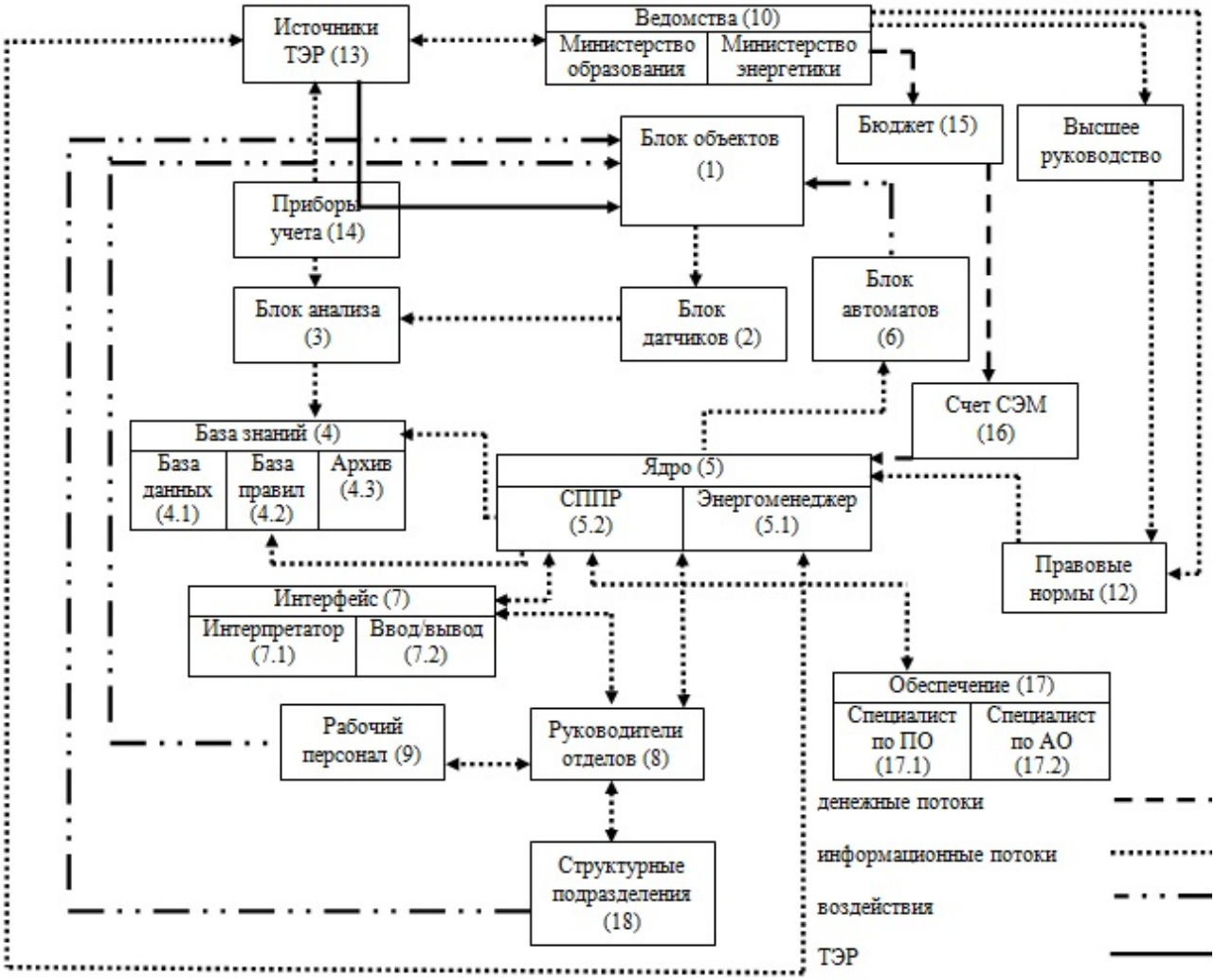


Рис. 1 - Концептуальная модель системы энергоменеджмента

Система энергоменеджмента является человеко-машинной объектно-ориентированной системой. Лицом, принимающим решение (ЛПР), в системе является энергоменеджер, а машинная составляющая представлена СППР. Обслуживанием СППР занимаются программист (17.1) и специалист по аппаратному обеспечению (17.2), в их задачи входит плановый осмотр, внесение изменений, отладка программ, технический осмотр, ремонт и тому подобное.

Система поддержки принятия решений - это компьютерная автоматизированная система, целью которой является оказание поддержки лицам, принимающим решение в сложных условиях, для полного и объективного анализа предметной деятельности.

Для разработки и реализации энергосберегающих мероприятий целесообразно классифицировать потребителей ТЭР, поскольку в зависимости от принадлежности

потребителя тому либо иному классу определяется структура, вид и содержание энергосберегающих мероприятий.

На концептуальном уровне все потребители ТЭР были объединены в один блок, названный блоком объектов (1), который моделирует часть окружающей действительности. На каждом из объектов находится определенный набор датчиков и автоматов, на схеме обозначенный как блок внутренних датчиков (2). Датчики собирают определенный вид данных об объекте (нахождение людей в помещении, работающее освещение или техника и т. п.) и направляют собранную информацию на блок анализа (3).

В блоке анализа (3) информация с датчиков обобщается, структурируется и передается в блок базы знаний (4).

Блок базы знаний (4) состоит из трех блоков: базы данных (4.1), базы правил (4.2) и архива (4.3). У каждого из этих блоков строго определенная функция: у базы данных (4.1) это принятие и структурирование данных, полученных с блока анализа; архив (4.3) сохраняет все полученные данные, что позволяет вести статистику; база правил (4.2) содержит правила поведения машинной части системы.

Далее возможно два варианта: первый вариант, когда в базе (4) знаний обнаружилась близкая по параметрам ситуация, и второй, когда аналога не обнаружилось.

В первом случае, когда при обработке информации в базе знаний (4) обнаружилась запись о ранее уже имеющей место ситуации и она аналогична или близка к текущему состоянию объекта. База знаний (4) передает информацию в СППР (5.2), которая обращаясь к базе правил (4.2), посредством внутренних алгоритмов анализирует входные параметры, просчитывает варианты и выделяет лучший из них. Получив результат, СППР (5.2) направляет команду на группу автоматов (6), которые напрямую воздействуют на объекты(1).

Таким образом, в этой ситуации участие людей не требуется, СЭМ получает и обрабатывает информацию и самостоятельно выносит эффективное решение.

В ситуации, когда в базе знаний (4) не обнаружилось ни одного похожего случая, СППР (5.2) не может принять самостоятельно решения о том или ином воздействии на объект (1). В этом случае СППР (5.2) так же анализирует с помощью внутренних алгоритмов ситуацию, сложившуюся на объекте (1) и выдает набор наиболее подходящих решений. Через интерфейс ввода–вывода (7), который с помощью интерпретатора (7.1) переводит машинный язык в понятный человеку, эта информация поступает к руководителю структурного подразделения – линейному руководителю (8) отвечающего за работу объекта или группы объектов. Ознакомившись с вариантами, предложенными СППР (5.2), руководитель принимает решение. Если реализация решения возможна с помощью установленных на объекте (1) автоматов (6), то руководитель (8) вводит необходимые команды в СППР (5.2) и та отправляет соответствующие команды автоматам (6). Если же реализация не возможна автоматами (6) то, внутри этого подразделения формируется команда (9), которая, в свою очередь, воздействует на объект (1). Если же проблемная ситуация выходит за должностные рамки линейного руководителя, то он направляет информацию выше по иерархии, где решения уже принимаются непосредственно энергоменеджером (5.1).

На всех этапах СППР (5.2) заносит данные в базу знаний (4), что делает систему динамичной и позволяет ей, подстраиваясь под любую ситуацию, минимизировать риски и увеличивать эффективность СЭМ.

Всю деятельность энергоменеджера и СЭМ в целом регламентирует блок правовых норм (12), являющийся совокупностью документов из двух источников:

1. Внешние документы: законодательные и нормативные акты, касающиеся выполняемой работы и другие. Задаются законодательным органом, блок ведомства (10), и являются обязательными.

2. Внутренние: устав предприятия, приказы и распоряжения руководства предприятия, должностные инструкции, правила внутреннего трудового распорядка и прочие. Задаются высшим руководством предприятия (11).

Обобщим все организации, предоставляющие ТЭР КнАГТУ, в один блок под названием источники ТЭР (13) и рассмотрим его далее. Деятельность этих организаций так е регламентирует государство (10) посредством законов, актов и так далее.

Энергоменеджер (5.1) заключает договора со сторонними организациями (13) на снабжение предприятия всеми видами ТЭР и контролирует их выполнение.

При поступлении ТЭР на предприятие их вид и объем фиксируются на приборах учета (14), эти данные одновременно фиксируются в блоках 13 и 3 и проходят выше описанный путь через базу знаний (4) и далее в СППР (5.2). СППР (5.2), взаимодействуя с базой знаний (4), сравнивает текущие показания с показаниями за прошлые периоды, прогнозирует расход ТЭР и формирует отчет для руководителей (8) и энергоменеджера (5.1). Благодаря этому энергоменеджер (5.1) может видеть детальную статистику поступления и потребления ТЭР за определенный период, проследить их потерю при транспортировке и наметить мероприятия по увеличению эффективности использования ТЭР.

Но СЭМ не ограничивается только технической стороной, одновременно с постоянным процессом улучшения техники и технологий в энергоменеджменте также особая роль выделена организационному вопросу, разработке и контролю над исполнением внутренних правил пользования потребителей ТЭР структурными подразделениями предприятия (19). Этой задачей занимается отдельная команда, руководитель которой напрямую взаимодействует с энергоменеджером (5.1).

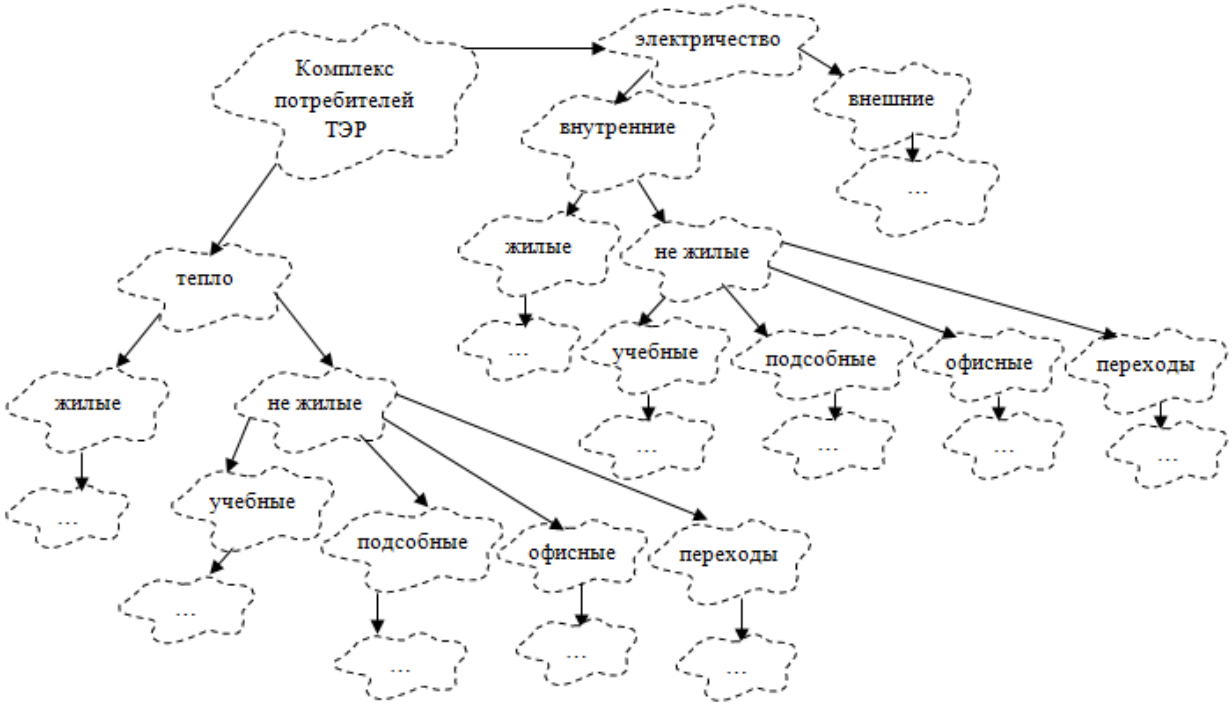


Рис. 2 – Классификация потребителей по контексту применения приемников

Для поддержания работоспособности системы требуются финансовые ресурсы. Если предприятие является государственным, то финансирование выделяет отвечающее за деятельность предприятие ведомство (10) и переводит денежные средства на счет предприятия (15). Если же предприятие не содержится на государственном обеспечении, а является частным, связь ведомства (10) с блоком бюджета (15) будет ограничиваться обратной связью. Финансовые средства перераспределяются внутри предприятия, и часть их направляется на поддержание и улучшение работы СЭМ. Эти средства переходят на внутренний счет СЭМ (16), которым, в свою очередь, управляет энергоменеджер, распределяя финансы среди структурных подразделений (8).

Поскольку количество потребителей на предприятии достаточно велико, с целью упрощения и ускорения деятельности СЭМ, предлагается классифицировать потребителей не только по типам приемников в отдельности, но и по контексту их применения и использования потребителей. Разделение по алгоритмам концентрирует внимание на порядке происходящих событий, а разделение по объектам придает особое значение агентам. Однако мы не можем сконструировать сложную систему одновременно двумя способами, тем более что эти способы, по сути, ортогональны. Целесообразно начать разделение системы либо по алгоритмам, либо по объектам, а затем, используя полученную структуру, попытаться рассмотреть проблему с другой точки зрения.

На рисунке 2 представлен фрагмент системы классификации объектов СЭМ на примере ФГБОУВПО КнАГТУ.

Например, аудитория 205 третьего корпуса является нежилым помещением, относящимся к классу лабораторных аудиторий, так как в ней присутствуют лабораторные стенды, на которых проводятся различные исследования.

На рисунке 3 представлен фрагмент системы классификации потребителей по типам приемников ТЭР, на примере ФГБОУВПО КнАГТУ.

Рис.3 - Классификация потребителей по типам приемников

Например, чугунная батарея (объект) относится к классу «чугунные», так как является радиатором с низким уровнем теплоотдачи (теплоотдача чугунного радиатора 100-200 ватт на секцию).

2. Пример работы СЭМ

Государство повысило норму по сокращению энергоемкости ФГБОУВПО КнАГТУ на будущий год. Ректор доводит новые требования до энергоменеджера и устанавливает цели и сроки. Энергоменеджер, руководствуясь рекомендациями СППР, с руководителями и ведущими специалистами подчиненных ему подразделений разрабатывают стратегию энергосберегающих мероприятий, прогнозируют возможные осложнения и рассчитывают финансовую сторону. Энергоменеджер предлагает на рассмотрение ректору план действий, ректор утверждает план и дает команду бухгалтерии выделить из бюджета КнАГТУ необходимую сумму. После одобрения плана и выделения средств энергоменеджер приступает к его реализации. По плану снижение энергопотребления на предприятии должно осуществиться за счет улучшения работы СППР, установки дополнительных датчиков на фасадах здания и утверждения норм и правил пользования потребителями ТЭР. Энергоменеджер назначает ответственного за закупку, установку и интеграцию в

существующую СЭМ новых датчиков пиргелиометров, которые улавливают интенсивность солнечного излучения. Ответственный за закупку и установку ведет переговоры с поставщиками, закупает приборы, собирает команду установщиков и контролирует процесс установки.

По завершению установки новых датчиков и прокладки необходимых кабелей для включения их в СЭМ за работу принимается отдел по обеспечению техническому и программному обеспечению. Программист изменяет алгоритм работы системы для программной интеграции нового оборудования, задает новые правила воздействий на автоматы. Специалист по аппаратному обеспечению улучшает аппаратную часть для ускорения работы системы с учетом дополнительных потоков информации. Энергоменеджер взаимодействуя с начальниками подчиненных ему подразделений тестирует работу СППР и обновленной СЭМ в целом. Одновременно ведется работа по составлению новых норм и правил пользования потребителями ТЭР. Дополнительная информация об интенсивности солнечного излучения позволит СППР более эффективно управлять автоматами, воздействующими на источники освещения или отопительную систему. Что приведет к уменьшению энергозатрат.

Выводы

Использование объектно-ориентированного подхода к проектированию СЭМ позволит справиться со сложностью и неоднородностью системы. Классификация и определение общих свойств позволяет ускорить процедуры идентификации объектов, объединять подобные объекты и создавать один алгоритм действия для группы (класса) объектов, что упростит реализацию и ускорит работу СЭМ.

Литература

1. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ / Г. Буч; пер. с англ. И. Романовского; под ред. Ф. Андреева – М.: Невский Диалект, 2000. – 359 с.
2. Министерство энергетики РФ // MINENERGO.GOV.RU: ежедн. интернет-изд. 2008. 22 июля. URL: http://minenergo.gov.ru (дата обращения 06.04.2013)

УДК 681.3

ИНТЕЛЛЕКТУАЛЬНАЯ СИСТЕМА ПРОЕКТИРОВАНИЯ ВЕБ-ПРИЛОЖЕНИЙ

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

В процессе проектирования декомпозицию системы разработки информационных систем обычно производят, исходя из максимальной независимости подсистем. Однако декомпозицию можно произвести в соответствии со взглядом на систему разных групп разработчиков (рис.1). Для каждой подсистемы указаны основные понятия ее предметной области.

Результатом взаимодействия всех этих подсистем и будет являться информационная система (ИС). Подобное деление проявляется как на уровне моделирования, когда каждая подсистема имеет свои группы описаний, представленные в соответствующих моделях, библиотеки модели и инструменты, так и на уровне реализации — каждая подсистема реализуется специфическими для нее программными компонентами. Результирующая ИС может быть рассмотрена как конфигурация взаимодействия групп объектно-ориентированных компонент. В свою очередь, каждая из групп компонент является результатом деятельности конкретной подсистемы проектирования во взаимодействии с другими подсистемами. Выявление совместно используемых понятий, моделей и т.п.,

принадлежащих разным подсистемам, позволяет сформировать единое информационное пространство взаимодействия этих подсистем, включающее общую, разделяемую модель данных. В этом случае различные описания системы не создаются в форме отдельных документов, а хранятся виде взаимосвязанных информационных единиц, готовых для объединения в той или иной форме. Таким образом, речь идет о хранении информации в виде базы фактов и базы знаний об организации этих фактов и доступе к этой информации с разнообразными запросами. Можно сделать вывод о том, что определяющим способом интеграции моделей является согласование понятий, определяющих представления моделей, что вводит необходимость семантического анализа как этапа проектирования системы. Инструментом проектирования в этом случае становится интеллектуальная система проектирования.

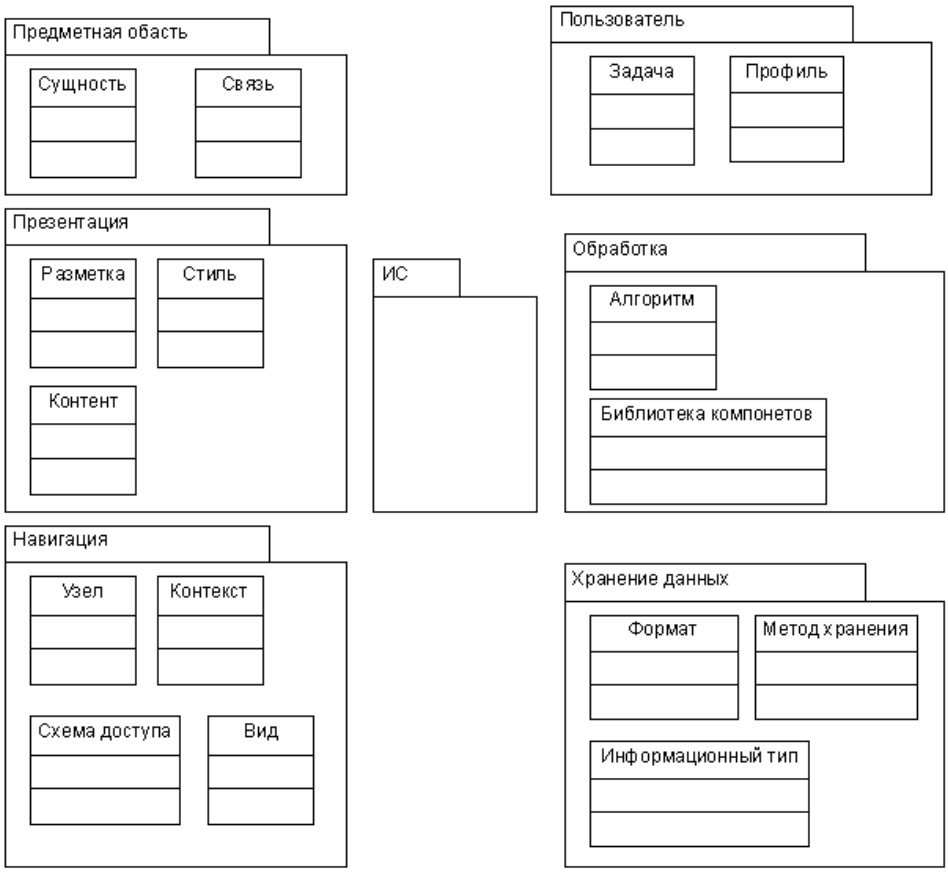


Рис.1 - Декомпозиция информационной системы

Под интеллектуальной системой обычно понимается любое программное средство, в архитектуре которого явно выделена база знаний [1]. Информационные системы, основанные на знаниях, строятся как специализированные оболочки над системой знаний сформулированных в терминах некоторой онтологии. В качестве формальной модели интеллектуальной системы можно использовать определение онтологической системы [2]:

$\sum^{O} = <O^{meta}, \{O^{d\&t}\}, \sum^{inf}>$,

где

O^{meta} — онтология верхнего уровня (метаонтология);

$\{O^{d\&t}\}$ — множество предметных онтологий и онтологий задач предметной области;

$\sum^{inf}$ — модель машины вывода, ассоциированной с онтологической системой $\sum^{O}$.

Под онтологией здесь понимается кортеж:

$O = <C, R, F>$

где:

C — множество концептов онтологии;

R — множество отношений между концептами онтологии;

F — множество функций интерпретации, заданных на концептах и отношениях.

Онтология включает в себя набор определений понятий предметной области и их взаимосвязей. Система проектирования должна интегрировать знания подсистем и позволять использовать эти знания при решении задач. Каждой из подсистем в общей модели проектирования можно сопоставить свою онтологию. Интеграция обеспечивается наличием метаонтологии, и все частные онтологии строятся на основе системы понятий, определенной в ней. Таким образом, можно представить модель информационной системы в процессе проектирования как кортеж онтологий:

OApp=<Meta, OApp, ODomain, OTask, ONav, OInfo, OComp, OGui, R>

где:

Meta — метаонтология;

OApp — онтология приложения;

ODomain — онтология предметной области;

OTask — онтология задач.

ONav — онтология навигации;

OInfo — онтология информационных элементов;

OComp — онтология компонентов;

OGui — онтология пользовательского интерфейса;

R — множество бинарных отношений между концептами и индивидуалами онтологий.

Каждая частная онтология отображается через отношения из R в некоторые подграфы других онтологий. Интеллектуальная система проектирования, являющаяся инструментальной оболочкой для такого информационного пространства, должна позволять хранить онтологию и знания каждой из подсистем проектирования, а также обеспечивать использование при решении задач тех информационных компонентов, которые в этом случае необходимы, то есть допускать интеграцию знаний и онтологий разных разделов области в рамках одного информационного ресурса. Таким образом, к информационной системе, основанной на знаниях, предъявляются требования:

1. Обеспечение возможности изменения знаний, структура которых определяется онтологией ПО.
2. Наличие средств модификации системы понятий предметной области.
3. Наличие возможности интеграции систем понятий из различных разделов ПО.

В этом случае знания, скрытые в кодах программного обеспечения, в компонентных моделях дизайна приложения и в других реализационных описаниях систем, будут представлены в виде сущностного, не зависящего от деталей реализации описания создаваемой системы. Таким образом, процесс проектирования в качестве своего контекста имеет сеть различных онтологий и решателей задач над нею. Решатели задач обеспечивают различные интерпретации сети онтологий, к которым следует отнести задачи генеративного программирования — полное или частично автоматизированное создание программных компонент. Обеспечение процесса проектирования и генерации компонентов требует методов и инструментов управления как для каждой отдельно взятой онтологии, так и для взаимного согласования указанных «универсальных» онтологий.

Для реализации такой системы проектирования нами был выбран подход, когда различным семантикам описания моделей ставятся в соответствие специализированные редакторы, реализующие эти семантики. Интеграция моделей производится с использованием редактора метаонтологии. Для этого в метаонтологии введены понятия, позволяющие определять виды деятельности, их результаты и исполнителей. Архитектура интеллектуальной системы проектирования представлена на рисунке 2.

Редактор навигационной модели использует онтологию навигации [3] для построения навигационной модели в терминах узлов и ссылок, определяя навигационный контекст и способ представления узла.

Редактор задач позволяет формировать требования к системе в терминах задач, процессов и исполнителей, связывая каждую задачу с элементами предметной области и

компонентом-исполнителем задачи.

Редактор пользовательского интерфейса позволяет определять шаблоны разметки веб-страниц приложения, моделировать визуальные элементы, определять для них источники данных, связывать элементы разметки шаблонов с конкретными визуальными элементами.

Редактор компонентов на основе модели предметной области позволяет определять схемы объектно-ориентированных классов и конкретизировать эти схемы с учетом программной системы, в контексте которой компоненты будут созданы.

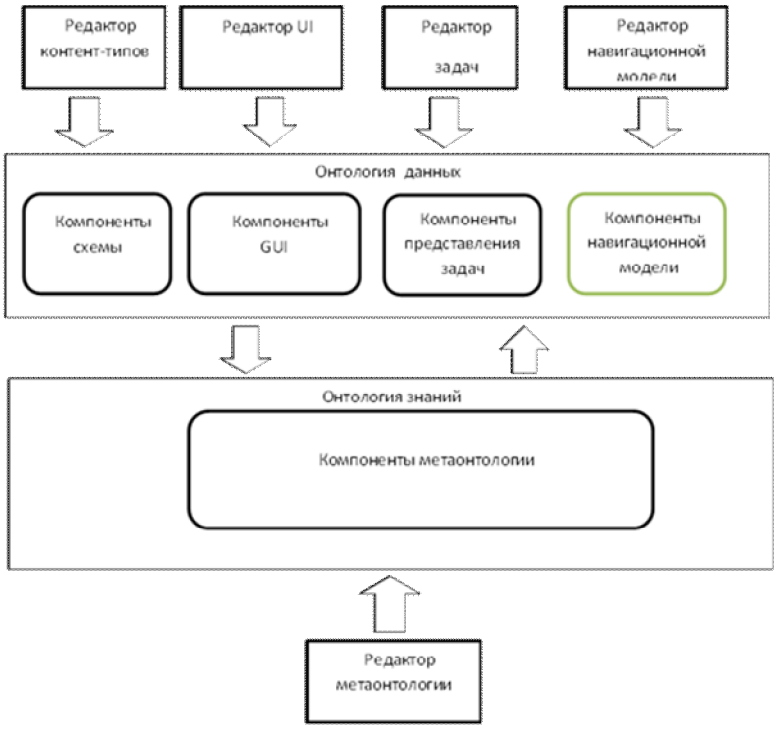


Рис. 2 - Архитектура интеллектуальной системы проектирования

Каждый редактор использует семантику предметной области, соответствующей подсистемы проектирования. Соответствующая модель, выраженная в терминах этой семантики и представленная выбранным способом визуализации, сохраняется в общей базе знаний системы проектирования. Хранение всех онтологий производится с использованием объектной базы данных [4]. Средством управления метаонтологией будет редактор концептуальных моделей [5], который будет строиться как манипулятор объектами, каждый из которых представляет собой ОО-реализацию элемента нотации метамодели [6].

Интеллектуальная система реализована на основе системы управления содержимым (CMS) Plone [7]. Выполнение инструментальной оболочки в виде веб-приложения позволяет обеспечить совместный удаленный доступ членам различных групп разработчиков, участвующих в проекте, причем распределение прав доступа в этом случае обеспечивается средствами CMS. И наконец, встраивание разрабатываемого комплекса в любой портал, также реализованный на Plone, предоставляет возможность управления порталом через редактирование соответствующих онтологий, а расширение возможностей портала — как через введение новых понятий в указанные онтологии, так и путем создания и подключения новых онтологических моделей.

Все редакторы имеют общую модель, UML-диаграмма модели классов редактора представлена на рисунке (Рис.3). В соответствии с этой моделью каждый редактор включает класс Model, который реализует интерфейс IContainer из API CMS Plone, определяющий поведение контейнера элементов с операциями управления этими элементами. Каждому редактору сопоставляется некоторая онтология, реализуемая внешним по отношению к редактору классом.

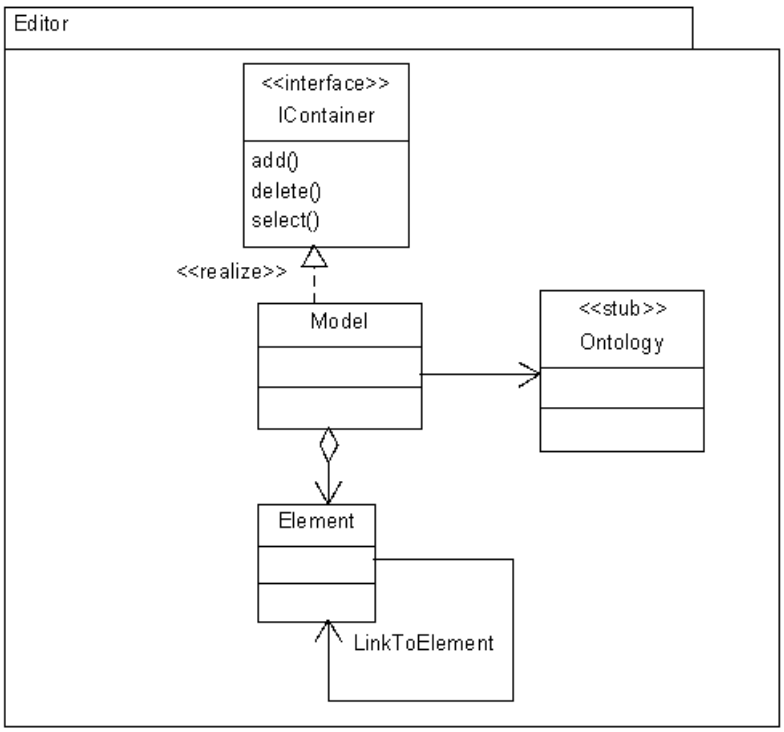


Рис. 3 - Общая диаграмма классов редактора

Модель включает в себя элементы, тип которых определяется связанной с редактором онтологией. Кроме того, элементы модели могут быть связаны между собой ссылками, при этом элементы могут принадлежать разным моделям. Таким образом, редактор представляет собой некоторый виртуальный контейнер, определенный на подграфе объектно-ориентированной семантической сети. Над этой сетью может быть определено неограниченное количество виртуальных редакторов различных типов.

Множество редакторов объединяются в разделы портала стандартными средствами управления контентом CMS Plone.

Литература

1. Клещев А.С., Шалфеева Е.А. Грибова В.В. Системы управления интеллектуальными Интернет-приложениями // Владивосток: ИАПУ ДВО РАН. – 2010. – с. 31.
2. Гаврилова Т.А., Хорошевский В.Ф. Базы знаний интеллектуальных систем. –СПб. Питер. 2000. –384 с.
3. Грегер С.Э. Проектирование и реализация навигационной онтологии сайта. // Объектные системы – 2012: материалы VI Международной научно- практической конференции (Ростов-на-Дону, 10-12 мая 2012 г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону:ШИ ЮРГТУ (НПИ), 2012. – с. 19-24.
4. Поршнев С.В. Грегер С.Э Совместное использование онтологической модели и объектной моделей при проектировании и реализации информационных WEB-систем // Естественные и технические науки. – №6. – 2011. – с. 461-468
5. Грегер С.Э. Редактор метамодели онтологической системы // Объектные системы – 2012: материалы VI Международной научно- практической конференции (Ростов-на-Дону, 10-12 мая 2012 г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ ЮРГТУ (НПИ), 2012. – с. 88-92.
6. Сковородин Е.Ю. Грегер С.Э. Построение онтологического портала с использованием объектной базы // Объектные системы - 2010: Материалы I Международной научно-практической конференции. – Ростов-на-Дону, 2010 г. – с. 74-78.
7. Грегер С.Э. Администрирование и интерфейс пользователя CMS Plone (монография) – Нижний Тагил: Федер. Агентство по образованию, ГОУ ВПО "УГТУ-УПИ им.первого Президента России Б.Н.Ельцина". Нижнетагил. технол. ин-т (фил.) НТИ(ф) УГТУ-УПИ. - 140с, 2009.

УДК 004.431.4

АВТОМАТИЗАЦИЯ РАБОЧИХ ПРОЦЕССОВ ПРЕДПРИЯТИЯ С ИСПОЛЬЗОВАНИЕМ COMINDWARE TRACKER

Трифонова Анна Александровна, студентка 5 курса, Ивановский государственный химико-технологический университет, Россия, Иваново, atri1991@mail.ru

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

Введение

Перед любым предприятием по мере его развития рано или поздно встает задача автоматизации управления бизнес-процессами. Решить эту задачу можно различными способами, каждый из которых будет иметь собственные преимущества и недостатки. Распространенным способом является создание решения на базе готовой программной платформы. Такой программной платформой могут послужить процессно-ориентированные системы, в частности workflow-системы, получившие распространение в последние несколько лет [1].

В данной статье рассматривается возможность автоматизации процесса выполнения заказа на полиграфическом производстве за счет создания решения в одной из существующих систем. В качестве такой системы была выбрана платформа Comindware Tracker [2]. Она представляет собой программное обеспечение для управления совместной работой, которое позволяет обеспечить коллективную работу различных отделов и команд. Система позволяет в визуальной среде создавать, автоматизировать и оптимизировать процессы, организовывать управление объектами (запросами, требованиями, заказами и др.), автоматически назначать задачи сотрудникам.

1. Постановка задачи

Пусть имеется следующая задача. На полиграфическом предприятии процесс формирования и выполнения заказа происходит следующим образом:

- клиент описывает заказ менеджеру;
- менеджер составляет паспорт заказа, содержащий следующую информацию: данные о заказчике, сроках выполнения, используемых материалах, технологиях, описание наносимых изображений и их цветность, данные о макетах и шаблонах, расчет стоимости единичного изделия и тиража, а также данные менеджера, принимавшего заказ;
- копия паспорта заказа отдается дизайнеру, разрабатывающему изображение и (или) макет;
- после создания макета и утверждения его заказчиком паспорт заказа передается инженеру по фотовыводу, который занимается подготовкой фотоформ;
- после того, как фотоформы подготовлены и проверены на корректность, паспорт заказа передается мастеру цеха;
- мастер цеха, основываясь на данных имеющихся у него на текущий момент паспортов, распределяет задания между работниками цеха (шаблонщиками, печатниками и работниками, отвечающими за дополнительные операции);
- после выполнения задания работником в паспорте делается пометка о выполнении заказа и он передается менеджеру;
- заказчик оплачивает заказ (если им не была осуществлена 100%-ая предоплата) и забирает готовые изделия.

В настоящий момент основным инструментом согласования действий между сотрудниками, участвующими в процессе выполнения заказа, является паспорт заказа, а именно, его бумажная копия. Передача необходимой для работы информации в таком виде

увеличивает временные издержки при информационном обмене между участниками процесса. В некоторых случаях такие задержки являются критичными – например, важные изменения в условиях заказа, оперативно не донесенные до непосредственных его исполнителей, может привести к браку большого тиража продукции, а, следовательно, к убыткам для фирмы.

Как отмечено в работе [3], такая проблема может быть связана с применением функционального подхода к управлению на предприятии. Внедрение решения, ориентированного на бизнес-процессы, а также обладающего гибкостью в плане изменения этих процессов, может стать толчком к переходу к процессному управлению. Кроме того, если предлагаемое решение будет достаточно простым, то предприятие сможет оперативно модифицировать его в соответствии с меняющимися требованиями бизнеса без привлечения внешних специалистов. Это, в свою очередь согласно выводам работы [4], снижает риск обособленности системы автоматизации от бизнес-процессов организации.

Таким образом, перед нами встает задача управления процессом выполнения заказа. Неэффективное управление может привести предприятие к потерям времени, заказов и прибыли. Поэтому важно, чтобы создаваемое решение предусматривало различные способы создания отчетности, позволяющей оперативно выявлять «узкие» места процесса. Разрабатываемое решение должно помочь усовершенствовать процесс оформления и обработки заказов за счет упрощения процессов согласования на каждом этапе выполнения заказа и снижения временных издержек, возникающих при передаче бумажной копии паспорта от одного участника процесса к другому. Кроме того, мы предполагаем, что такое решение предприятие может создать самостоятельно на базе готовой системы, без привлечения внешних специалистов.

На первом этапе работы над задачей нами был проведен анализ предметной области и существующих в ней бизнес-процессов. На втором этапе на основе описанного бизнес-процесса мы попытались создать работоспособное решение, удовлетворяющее требованиям заказчика, в системе Comindware Tracker. С результатами работы мы готовы ознакомить читателей в настоящей статье.

2. Предлагаемое решение

Система Comindware Tracker оперирует такими понятиями, как шаблон, рабочий процесс, группа и рабочая область, поэтому далее приведено их краткое освещение (рис. 1). Более подробную информацию можно найти в документации по системе [5].

Создание любого объекта в системе Comindware Tracker происходит на основе шаблона. Следовательно, то, как будет выглядеть объект и каким будет его поведение, зависит от настройки параметров шаблона. Она, в свою очередь, сводится к созданию полей шаблона (поля определяют информацию, которую будет нести в себе объект), настройке формы (форма определяет способ отображения полей шаблона) и настройке рабочего процесса (рабочий процесс определяет жизненный цикл созданного объекта). Поскольку нам важно не только создать заказ в системе в виде некоторого статичного документа, но и контролировать процесс его выполнения, то заложенная в системе концепция рабочих процессов представляется нам хорошим инструментом для достижения поставленных целей.

По сути, рабочий процесс представляет собой конечный автомат или диаграмму состояний, описывающую возможные последовательности переходов объекта из одного состояния в другое. В терминах системы Comindware Tracker состояние - это шаг, на котором находится объект в рабочем процессе. Для связи шагов и перемещения объекта с предыдущего шага на следующий используются переходы. Срабатывание перехода зависит не от выполнения определенных условий (исключается возможность проверить с помощью логического условия возможность перехода), а только от наступления события. Событием, инициирующим переход объекта, может являться задача (также возможен «ручной» перевод объекта на следующий шаг). Задача выступает в качестве поручения для какого-либо сотрудника и закрепляет ответственность за выполнение им некоторой работы. В свою

очередь задача, как и объект, может переходить из одного состояния в другое. Разница здесь состоит в том, что в системе для задач существует заданный набор состояний и изменить его нельзя.

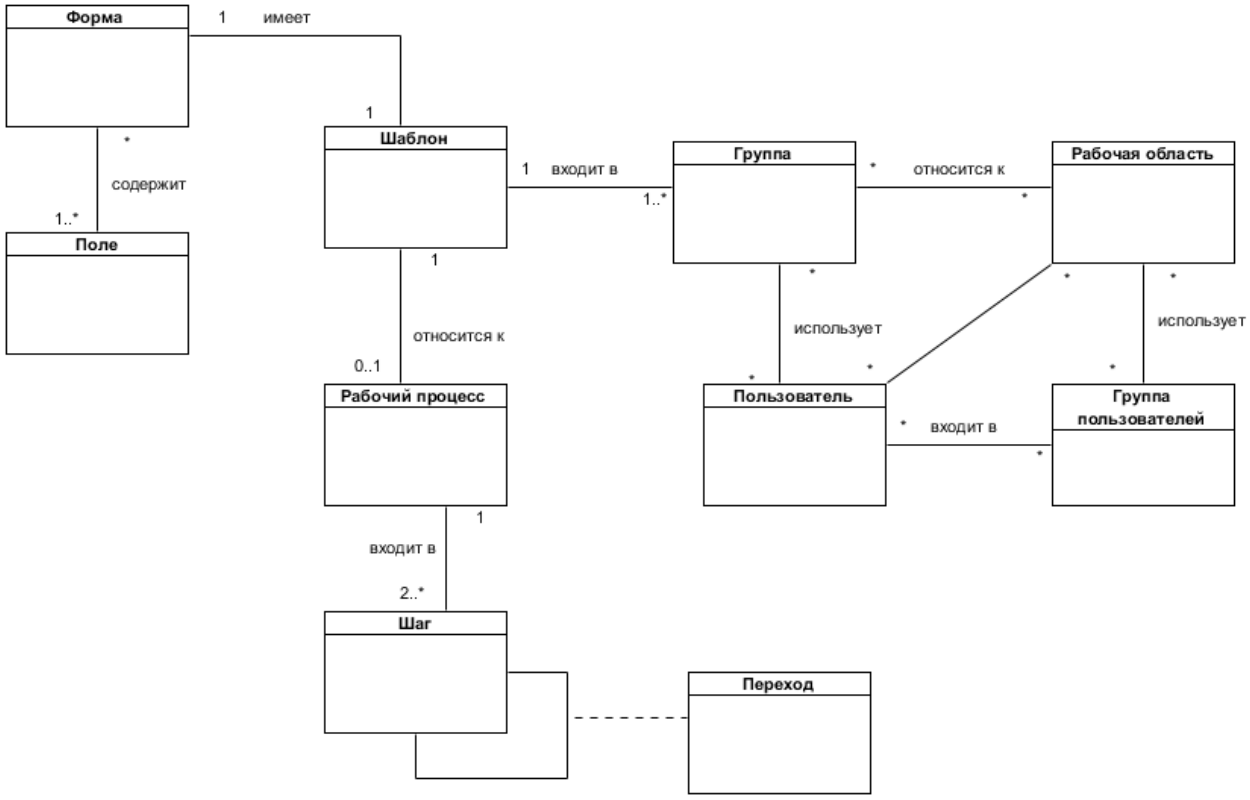


Рис. 1 – Структурная модель понятий системы Comindware Tracker

Для объединения однотипных объектов (созданных на основе одного шаблона) существуют группы. Также группы позволяют настроить набор разрешенных действий (уровни доступа) для пользователей и групп пользователей.

Созданную группу можно добавить в рабочую область. Рабочая область представляет собой объединение групп задач и объектов, совместно используемых группами пользователей. С помощью рабочей области можно создавать отдельные рабочие среды для различных отделов или проектов в рамках организации. Несомненным плюсом является то, что вместе с системой поставляются примеры настроенных рабочих областей, включающих в себя готовые шаблоны, рабочие процессы и группы объектов. Это позволяет быстрее освоить принципы работы с системой и упрощает процесс ее настройки.

Сформировав некоторое представление об основных идеях, заложенных в Comindware Tracker, мы попытались настроить систему в соответствии с условиями поставленной задачи.

Как было сказано выше, основным источником информации для ответственных за выполнение заказа лиц является паспорт заказа. Поэтому было решено создать шаблон для заказа, которого было бы достаточно для занесения в систему всей той информации, что несет в себе паспорт заказа. В результате для шаблона были созданы поля, содержащие информацию о заказчике, об особенностях заказа и его стоимости.

Настройка шаблона, прежде всего, предполагает создание необходимых полей. Система Comindware Tracker содержит большой список системных полей, которые мы можем использовать по умолчанию, и, как показывает практика, иногда этого вполне достаточно - в общем случае любому объекту необходимы такие поля как, например, идентификатор или описание. Также полезным оказалось наличие такого системного поля как "Создан" - с его помощью мы всегда сможем идентифицировать менеджера, добавившего заказ в систему.

Для того чтобы отразить особенности предметной области, нами были созданы собственные поля согласно модели паспорта заказа, представленной на рис. 2. Система

позволяет работать с различными типами полей (более подробно о них можно узнать в документации по системе). По большей части нами использовались такие типы полей как текстовое, числовое, а также список значений.

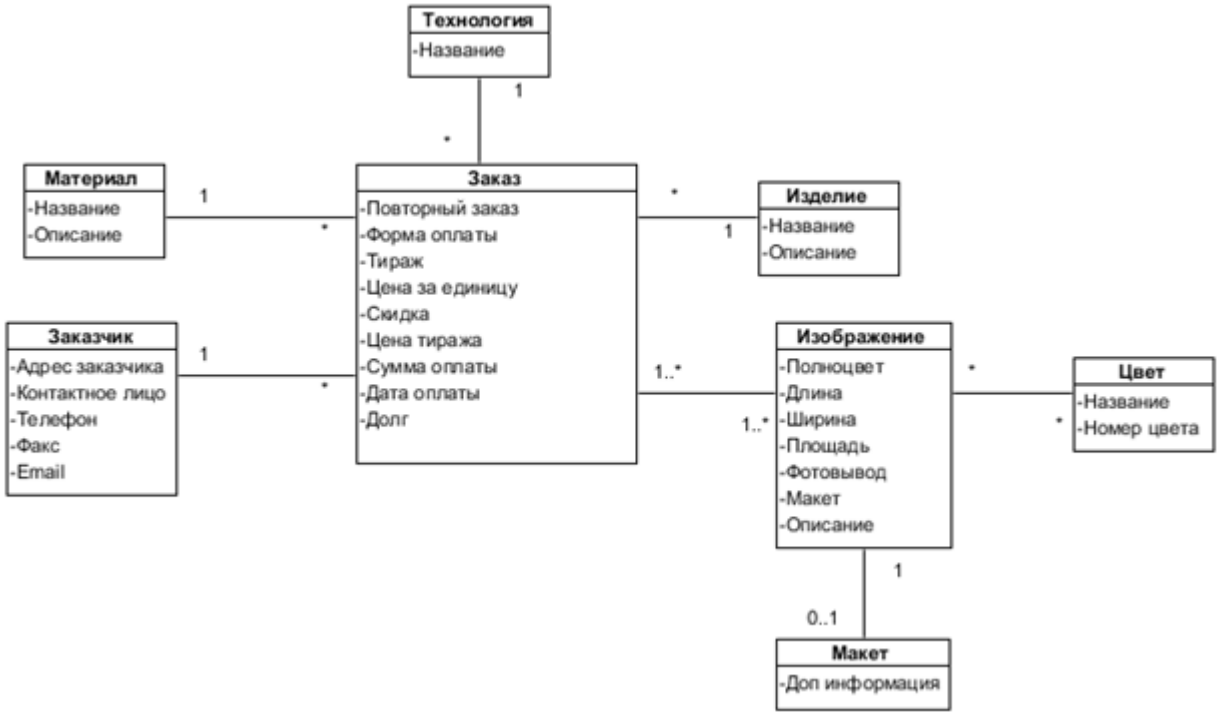


Рис. 2 – Диаграмма классов паспорта заказа

Список значений был применен для фиксированных наборов значений (например, набор возможных форм оплаты - взаимозачет, предоплата) и в качестве замены полей булевского типа, которые система не поддерживает. Неудобства также вызвало отсутствие возможности использования чисел с плавающей запятой. Comindware Tracker предоставляет собственный язык выражений для вычисления значений полей. В формулах используется содержимое других полей, поэтому нам показалось логичным попробовать применить такую возможность для вычисления цены тиража или площади изображения. Но, как было сказано выше, в системе отсутствует тип, поддерживающий дробные числа (а именно такой может быть цена со скидкой или площадь изображения), поэтому в этом случае от использования вычислимых полей пришлось отказаться.

Тем не менее, вычислимые поля все же нашли применение в нашем решении. На практике часто возникает ситуация, что клиент обращается в организацию более одного раза и, следовательно, нет необходимости вторично заносить в систему уже имеющиеся в ней данные о клиенте. Поэтому был создан шаблон объекта "Заказчик", содержащий в себе все необходимые поля с информацией о нем (адрес, контактное лицо, телефоны и т.д.). В шаблоне объекта для заказа были созданы поле "Заказчик" с типом "Ссылочное" (оно ссылается на объект "Заказчик") и поля, аналогичные используемым в шаблоне "Заказчик", только с заданным свойством "Вычисляемое" и некоторым выражением поля. Таким образом, если при создании заказа в поле "Заказчик" будет выбрано значение из списка занесенных в систему, то настроенные поля заполнятся автоматически после сохранения заказа. Такое поведение системы, несомненно, облегчает работу менеджеров при приеме заказов от постоянных клиентов.

Система Comindware Tracker обладает развитыми средствами для настройки формы ввода информации об объекте. Поэтому, когда созданы все необходимые поля, редактирование формы не представляет особой сложности - нужно просто добавить поля на форму (это делается перетаскиванием их мышью) и разместить их так, как мы считаем нужным.

Заказ обладает ясно выраженным жизненным циклом, который мы можем описать посредством рабочего процесса. Как уже было сказано, в рассматриваемой нами организации имеет место ситуация повторного обращения клиентов. Однако, существуют и другие условия, при воссоздании которых изменится рабочий процесс (например, у клиента есть готовый макет, следовательно, услуги дизайнера ему не требуются; клиент хочет заказать дополнительный тираж, следовательно, можно сразу перейти к непосредственному выполнению заказа без предпечатной подготовки). Для таких ситуаций мы решили создать дополнительные шаблоны и, следовательно, рабочие процессы, отличающиеся друг от друга отсутствием определенных стадий предпечатной подготовки (рис. 3).

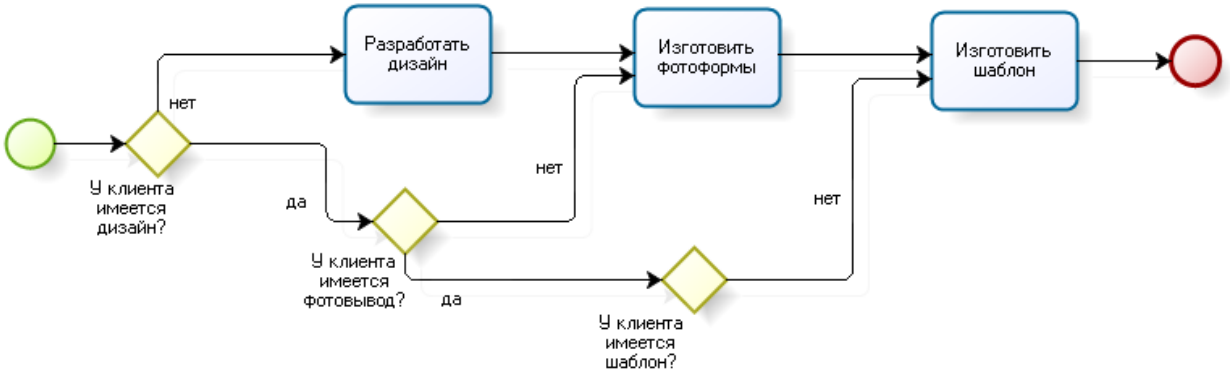


Рис. 3 – Модель подпроцесса “Предпечатная подготовка”

В результате мы получили 4 шаблона рабочих объектов, реализующих собой 4 возможных варианта заказа: "типичный" заказ, которому необходимы все три стадии предпечатной подготовки (разработка дизайна, изготовление фотоформы, изготовление шаблона) (схема рабочего процесса представлена на рис.4); заказ без дизайна; заказ без фотовывода; заказ без шаблона.

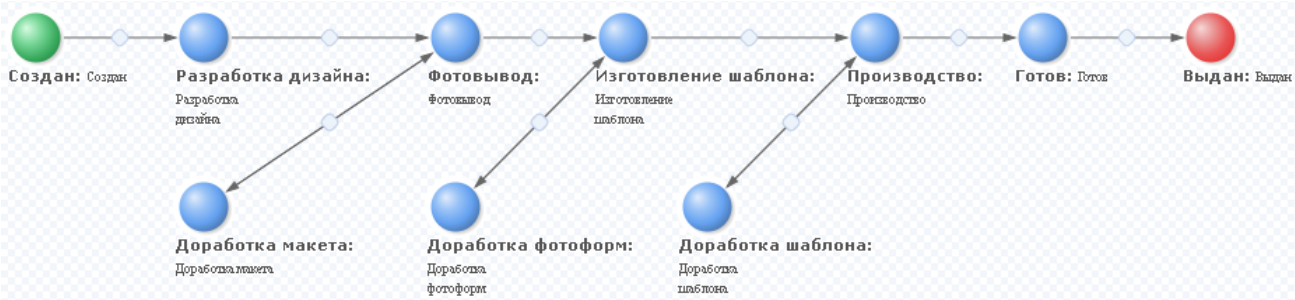


Рис. 4 – Схема рабочего процесса для шаблона объекта “Типичный заказ”

Для автоматического создания задач работникам в процессе выполнения заказа было настроено автоматическое создание задач при переходе объекта на любой шаг, кроме начального и конечного. Как говорилось ранее, задачи создаются для закрепления ответственности за определенный шаг за сотрудником. С одной стороны, если у нас по одному пользователю для каждой имеющейся роли, то мы просто укажем конкретного пользователя, для которого будет создаваться задача. Но с другой стороны такая ситуация далека от реальности, поэтому мы решили проблему следующим образом – задача назначается определенному пользователю, который выступает в роли руководителя (у дизайнеров это может быть, например, главный дизайнер, в цехе это мастер цеха и т.д.), и уже он переадресует эту задачу своему подчиненному. Возможно, что такая схема в реальности покажется малоэффективной, но следует понимать, что у любой системы могут быть свои нюансы, под которые придется подстраиваться.

Рассматриваемое нами в качестве примера полиграфическое производство, как и любое другое предприятие, обладает некой организационной структурой. Ответственность за тот или иной участок процесса закреплена за людьми с определенными должностями, и важно учесть этот факт при настройке системы. В соответствии с ролями, исполняемыми работниками организации в процессе выполнения заказа, было выделено 5 групп

пользователей: менеджеры, дизайнеры, инженеры по фотовыводу, мастера цеха, работники цеха. Также для того, чтобы логически разделить непосредственных исполнителей заказа (работников цеха) и работников, занимающихся работой с клиентами и предпечатной подготовкой, были созданы две рабочие области – "Офис" и "Цех".

Естественно, что нам необходимо не только разделение пользователей на определенные категории, но и назначение им определенных привилегий и ограничений. Например, работнику цеха не нужна возможность создавать заказ в системе, т.к. созданием заказов занимаются только менеджеры. Такие и подобные ограничения мы реализовали путем настройки уровней доступа к группам объектов, а также добавлением группы объектов к нужной рабочей области (это необходимо для отображения групп). Для того чтобы реализовать описанное выше ограничение, мы установили уровень доступа "Участники" к группам объектов, связанных с шаблонами для создания заказа, для группы пользователей "Менеджеры" и добавили группы объектов к рабочей области "Офис". Это значит, что все пользователи, входящие в группу "Менеджеры", смогут создавать, изменять и просматривать заказы. Поскольку для группы "Работники цеха" мы не устанавливали уровней доступа, то члены этой группы не смогут создавать и изменять заказы. При этом работники могут выполнять задачи в рамках рабочего процесса и, таким образом, переводить заказ из одного состояния в другое.

Для получения оперативной информации о движении заказа и текущей стадии его выполнения нам необходимо использование различных отчетов. Система Comindware Tracker предлагает два способа их реализации: в виде списков, позволяющих группировать объекты или задачи по определенным критериям, а также в виде панелей мониторинга, представляющих собой набор настраиваемых диаграмм и графиков (виджет в терминах системы) для отображения информации в удобном графическом представлении. Списки и панели мониторинга могут быть как доступными только пользователю, создавшему их, так и общими. Это позволяет один раз настроить необходимые отчеты и предоставлять к ним доступ определенным пользователям.

Был создан список «Все заказы», агрегирующий в себя заказы, созданные по 4 настроенным шаблонам; он доступен только тем группам пользователей, которое имеют разрешение на просмотр заказов в соответствующих группах объектов. Такой несложный отчет позволяет менеджерам увидеть, на какой стадии выполнения находится тот или иной заказ, кто ответственен за его выполнение на этой стадии, а также насколько быстро выполняются отдельные заказы и заказы в целом.

Виджеты создаются на основе списков. Каждый виджет может содержать данные из одного или нескольких списков. Для отображения актуальной информации об имеющихся заказах, а также о выполнении заказов конкретных заказчиков была создана панель мониторинга «Статистика по заказам», содержащая два виджета – круговую диаграмму «Все заказы по состояниям» и линейчатую диаграмму «Состояние заказов (по заказчикам)». Обе диаграммы строятся на основе данных из созданного нами ранее списка «Все заказы» и фактически представляют собой удобный способ отображения необходимой менеджерам и руководителям информации.

Выводы

Таким образом, высказанное нами ранее предположение о возможности создания полноценного рабочего автоматизированного решения без привлечения внешних специалистов во многом подтвердилось. Создание нового решения или адаптация имеющегося в Comindware Tracker не вызывает особых трудностей. Но, сталкиваясь с простотой и доступностью использования готовой системы, приходится жертвовать сложностью реального процесса и идти на определенные допущения.

Литература

1. А. Резниченко. Системы workflow/BPM - проблемы роста // PCWeek/RE, № 29, 2005. URL: http://www.pcweek.ru/idea/article/detail.php?ID=70973 (дата обращения: 12.04.2013).

2. Приложение для управления совместной работой – Comindware [сайт]. URL: http://www.comindware.com/ru (дата обращения: 12.04.2013).
3. Д. Пинаев. Процессное управление: в чем сила? // БОСС, № 3, 2012. URL: http://www.bossmag.ru/archiv/2012/boss-03-2012-g/protsessnoe-upravlenie-v-chem-sila.html (дата обращения: 12.04.2013).
4. Ю. Вагнер. BPM без бизнеса // Открытые системы, № 2, 2012. URL: http://www.osp.ru/os/2012/02/13014103 (дата обращения: 12.04.2013).
5. Руководство по продуктам Comindware [сайт]. URL: http://www.comindware.com/ru/support/resources (дата обращения: 12.04.2013).

УДК 004.413

ФРЕЙМВОРК ИНТЕГРАЦИИ ПОИСКОВОЙ МАШИНЫ В ИНФОРМАЦИОННЫЕ СИСТЕМЫ

Крылов Александр Юрьевич, студент, Ивановский государственный химико-технологический университет, Россия, Иваново, qblook@gmail.com
Галиаскаров Эдуард Геннадьевич, к.х.н., доцент, доцент, Ивановский государственный химико-технологический университет, Россия, Иваново, galiaskarov@isuct.ru

Любая современная информационная система (ИС) работает с большим количеством информационных ресурсов, которые представлены в виде баз данных, текстовых документов различного формата, электронных таблиц, адресных книг, профилей пользователей, лог-файлов, изображений, аудиофайлов и т. п.

С целью облегчения работы пользователей с системой, разработчики снабжают системы возможностями информационного поиска по используемому в ней контенту. При этом, как отмечено в работе [1], разные формы представления информационных ресурсов требуют организации различных форм поиска по ним.

Реализация поиска в современных ИС зачастую представляет собой отдельный функциональный модуль системы, направленный на решение данной задачи. Качество такого модуля во многом определяется результатами его работы: гибкостью, удобством, возможностями масштабирования, скоростью работы и т. д. С целью решения этих задач, разработчик прибегает к самостоятельной реализации таких модулей, которые, в свою очередь [2,3]:

- вынуждают разработчиков ИС повторно разрабатывать подсистему поиска (временные затраты) для каждой системы и типов документов;
- обладают низкой гибкостью и слабым функционалом, в силу того, что не являются основной целью разработки информационной системы;
- требуют технической поддержки и сопроводительной документации в течение всего процесса функционирования (эксплуатации);
- содержат различные ошибки в концепции, архитектуре или программной реализации;
- обладают недостаточной документацией;
- слабо стандартизированы;
- плохо пригодны к модификации, поскольку являются побочным результатом построения ИС.

Рассмотрев в качестве примера принципы функционирования документно-ориентированных систем [4], отметим, что реализация подсистемы поиска может представлять собой достаточно типичную задачу. Такое предположение рождает предпосылки того, что возможна некоторая стандартизация поисковых подсистем, их унификация.

В результате возможно создание некоторого универсального средства (инструмента), предназначенного для реализации поиска в целевой информационной системе, которое

значительно облегчило бы разработку поисковых подсистем и их последующую интеграцию в целевую ИС.

Создание такого инструмента позволит обеспечить:

- исключение необходимости повторно разрабатывать подсистему поиска для каждой новой системы;
- наличие единой и полной, а также регулярно обновляемой документации;
- отсутствие наиболее частых и распространенных ошибок, возникающих в процессе разработки и реализации поисковых систем;
- наличие технической поддержки.

Цель разработки данного инструмента (фреймворка) сводится к облегчению процесса разработки и интегрирования поискового функционала в информационную систему.

Для достижения поставленной цели необходимо решить следующие задачи:

- разработать единую архитектуру поисковых подсистем;
- в соответствии с разработанной архитектурой реализовать фреймворк, предоставляющий разработчику набор готовых решений для реализации этой подсистемы.

Для описания архитектуры и принципов функционирования эталонной поисковой подсистемы введем следующие понятия предметной области:

- *документ* — целевая информационная единица, искомая пользователем системы. Может представлять собой текстовой документ, изображение, аудиотрек, электронную таблицу, профиль пользователя, статью, главу книги и т.д., имеющие определенный набор мета-информации;
- *коллекция* — набор документов, объединенных каким-либо общим признаком (типом файла, источником, датой и т. д.), в нашем случае — типом файла, и по которым осуществляется поиск;
- *поисковой запрос* — набор параметров (инструкция), с помощью которых осуществляется отбор документов;
- *тело запроса* — набор параметров (атрибутов), описывающих поисковый образ искомых документов (условия отбора). Может представлять собой текст, хеш-сумму, дату, размер файла и т. д. - в зависимости от типа искомых документов. Является частью поискового запроса;
- *клиентская часть* — часть поисковой подсистемы, выполняемая программным обеспечением, установленным на рабочей станции пользователя системы;
- *серверная часть* — часть поисковой подсистемы, выполняемая программным обеспечением сервера системы;
- *пользователь системы* — лицо, составляющее и инициирующее выполнение поискового запроса, а также использующее результаты его выполнения;
- *поисковый движок* - подсистема, реализующая поиск и выборку документов по поисковому запросу в соответствии с заложенной в нее поисковой моделью (поисковым алгоритмом);
- *программный каркас (фреймворк)* — набор программных элементов и правил их использования для реализации поискового функционала.

Для создания целевого программного каркаса необходимо выделить определенные сходства поисковых подсистем, реализованных в рамках различных проектов (информационных систем).

Рассмотрим пример взаимодействия пользователя с поисковой подсистемой:

- пользователь формирует поисковой запрос, описывающий характеристики (параметры) интересующих его документов, предоставляя системе поисковой образ запроса искомого документа (ПОЗ), и инициирует выполнение запроса;
- система производит поиск и выборку документов, соответствующих тем или иным способом организованному поисковому образу документа (ПОД), и предоставляет

результаты пользователю;

- пользователь работает с результирующей выборкой документов, возможно, уточняя ПОЗ искомых документов.

Анализируя приведенный сценарий, можно утверждать, что типичная поисковая подсистема должна представлять собой реализацию:

- средств формирования поискового запроса;
- средств передачи поискового запроса поисковому движку;
- средств выполнения бизнес-логики и поиска;
- средств представления результатов поиска и работы с ними в удобной форме.

Обычно поисковая система оперирует документами определенного типа. Если же абстрагироваться от типа искомых документов, процессы функционирования поисковых систем (являющиеся частью других систем или же самостоятельными) можно свести к единому сценарию, обеспечивая тем самым возможность создания унифицированного инструмента для реализации данного функционала.

Данный инструмент должен представлять определенную программную архитектуру, которой должна соответствовать поисковая система. Рассмотрим такую архитектуру на примере ИС, реализованных в виде web-приложения. В соответствии с этим, подсистема поиска, разработанная при помощи данного фреймворка, разделяется на две части: клиентскую и серверную. В данном случае, клиентская часть предоставляет пользователю средства формирования поискового запроса и просмотра результатов поиска (работы с ними). Серверная часть — реализует средства обработки (выполнения) пользовательского поискового запроса.

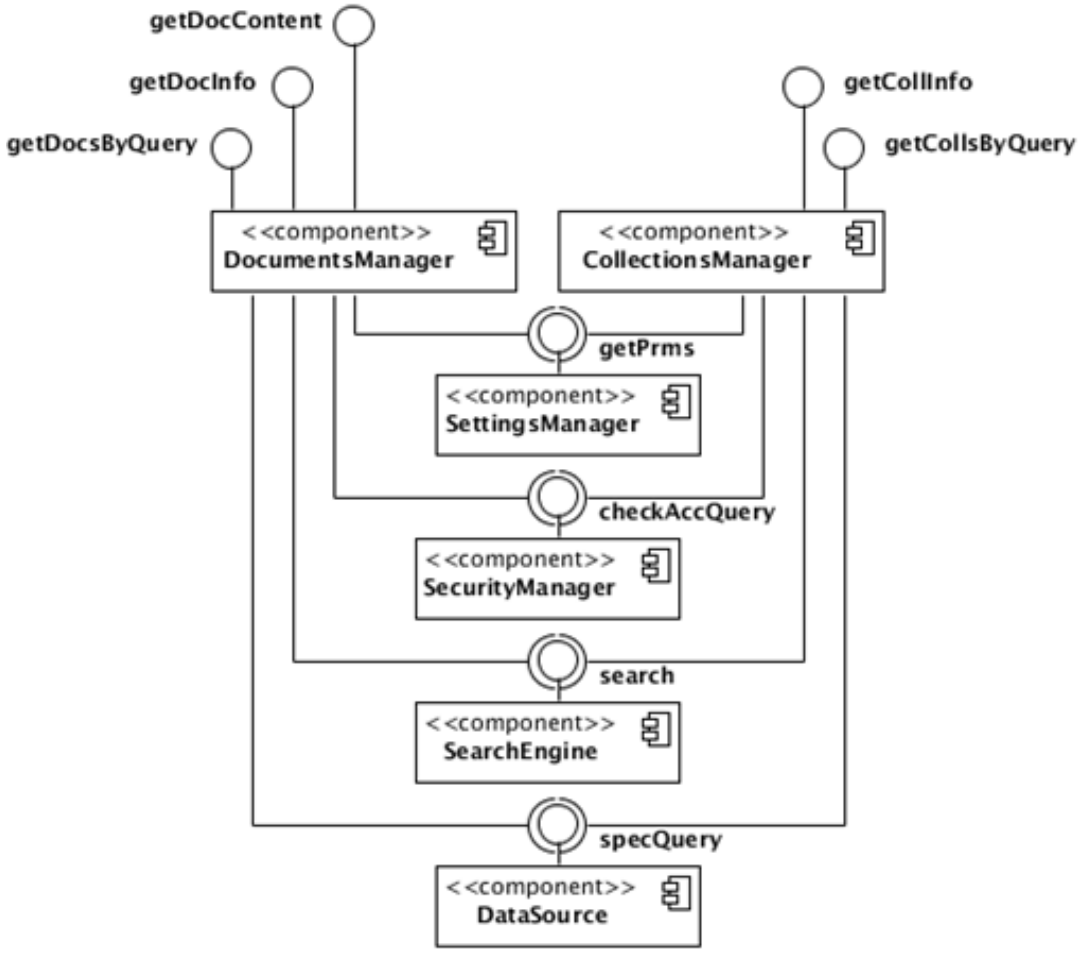


Рис. 1 – Компонентная структура серверной части подсистемы поиска

Данная часть системы содержит следующие компоненты:

- **DocumentsManager** - компонент, направленный на работу с документами, для которых производится поиск. Организует проверку прав доступа к определенным коллекциям, поиск документов, загрузку метаинфомрации документов и т.д.;
- **CollectionsManager** - компонент, направленный на работу с коллекциями, по которым производится поиск документов;
- **Query —** структура. Содержит в себе поисковой образ документа, а так же некоторые параметры, касающиеся сортировки результатов поиска, их максимальное количество, тип соответствия документа поисковому образу и т. д.;
- **SearchEngine** - компонент, инкапсулирующий в себя логику поиска, динамически подгружая соответствующий коллекции скрипт-обработчик, предназначеный для поиска по указанному типу документов и типу поискового запроса;
- **DataSource** - компонент, являющийся источником контента. Представляет собою прослойку между подсистемой поиска и базой данных ИС, частью которой он является;
- **SecurityManager** - компонент, отвечающий за доступ текущего пользователя к той или иной информации (коллекциям, документам);
- **SettingsManager** - компонент, содержащий в себе параметры/настройки. Такие, как лимит документов-результатов поиска по умолчанию, степень соответствия по умолчанию, сортировка по умолчанию и т. д.

На рис.1 представлена структура серверной части подсистемы поиска. Клиентскую часть подсистемы поиска должна содержать в себе следующие компоненты.

Средства формирования поискового запроса. Представляют собой элементы системы генерации пользовательского интерфейса, предназначенного для редактирования описания поискового образа документа;

Средства передачи пользовательского поискового запроса к серверной части системы. Представляют собой программный слой между элементами логики клиентской части системы и API-интерфейсами, предоставляемыми серверной частью системы, предназначенные для инкапсуляции формата передачи данных (поискового запроса, результатов поиска, параметров функционирования и т.д.) между клиентской и серверной частью системы;

Средства работы с результатами поиска. Представляют собой набор элементов, предназначенных для обеспечения пользователю поисковой системы возможности просмотра и работы с документами-результатами поиска.

Архитектура разрабатываемой системы должна обеспечивать распределенность, слабую связанность, а так же взаимозаменяемость компонентов системы, оснащенных стандартизированными интерфейсами. Данные задачи позволяет решить следующий подход к разработке решения:

- использование сервис-ориентированной архитектуры [5];
- разработка решения в соответствии со стандартом REST [6].

Преимуществами такого подхода будут являться:

- полная независимость клиентской части от серверной (и наоборот), что позволяет производить поиск документов в информационной системе не только с web-странички, но и из других, смежных систем, делающих поисковые запросы к целевой системе (для чего достаточно лишь обеспечить в смежной системе поддержку используемого поисковой системой формата передачи данных);
- независимость серверной части системы от механизмов поиска, специфических для каждого из используемых в системе форматов документов;
- независимость серверной части системы от типа и формата используемых информационных ресурсов.

Рассмотрев разработанную для поисковой подсистемы архитектуру, можно заметить, что реализация данных компонентов (классов) значительно облегчает разработчику

реализацию и интегрирование поисковых возможностей в информационную систему. При этом разработанное решение обладает:

- масштабируемостью — при расширении функционала поисковой подсистемы возможна модификация существующих и добавление новых классов (внесение изменений в программную архитектуру подсистемы), а также изменение числа информационных ресурсов, задействованных в информационной системе — добавление нового хранилища данных, распределение единой базы данных в кластер и т.д.. Таким образом, решение поддерживает как вертикальное, так и горизонтальное масштабирование;
- надежностью — в силу простоты программной архитектуры решения, реализованная при помощи разрабатываемого фреймворка поисковая система (подсистема) значительно легче отлаживается, что, в свою очередь, позволяет избежать множества ошибок;
- конфигурируемостью — параметры функционирования системы реализуются в общедоступных форматах (например, xml, json и т. д.), а также доступны для изменения при помощи модификации соответствующих конфигурационных параметров. Такой подход делает реализованную при помощи данного инструмента систему гибкой и конфигурируемой.

Литература

1. Д. Баженов. Архитектура поисковых систем // Блог Дениса Баженова [сайт]. URL: http://bazhenov.me/blog/2013/01/08/search-architecture.html (дата обращения 14.04.2013).
2. Разработка собственной поисковой системы с помощью PHP. URL: http://www.ibm.com/developerworks/ru/library/os-php-sphinxsearch/ (дата обращения 14.04.2013)
3. Google AdSense [Материал из Википедии — свободной энциклопедии]. URL: http://ru.wikipedia.org/wiki/AdSense (дата обращения 14.04.2013).
4. Документальные информационные системы // ПИЭ.Wiki [сайт]. URL: http://wiki.mvtom.ru/index.php/Документальные_информационные_системы (дата обращения 14.04.2013).
5. Сервис-ориентированная архитектура // CIT Forum [сайт]. URL: http://citforum.ru/internet/webservice/soa/ (дата обращения 14.04.2013).
6. Знакомьтесь, архитектура REST // Блог HTML-Templates.Info [сайт]. URL: http://html-templates.info/view_blog.php?id=6 (дата обращения 14.04.2013).

УДК 004.652.5

МОДЕЛИ И МЕТОДЫ ПОСТРОЕНИЯ СИСТЕМЫ OLAP ДЛЯ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ БАЗ ДАННЫХ[1]

Фисун Николай Тихонович, д.т.н., проф., зав. кафедры интеллектуальных информационных систем, ЧГУ им. П. Могилы, г. Николаев, Украина, mykola.fisun@gmail.com, ntfis@kma.mk.ua
Горбань Глеб Валентинович, асп. кафедры интеллектуальных информационных систем, ЧГУ им. П. Могилы, г. Николаев, Украина, bobsley2006@ukr.net

Введение

В настоящее время в системах поддержки принятия решений (СППР) важное место занимают технологии оперативного (OLAP) и интеллектуального (Data Mining) анализа данных [1, 2]. Данные технологии уже реализованы в некоторых коммерческих системах управления базами данных (СУБД), таких как Microsoft SQL Server, Oracle и других, которые представляют собой реляционные СУБД.

Реляционные базы данных (БД) получили большой успех и на данный момент

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

являются самыми распространенными. Это обосновано тем, что данные в них сохраняются в табличном формате и доступны для каждого, кто знаком со структурированным языком запросов (SQL). Однако со временем информационные системы предъявляют такие требования к обработке данных, с которыми реляционные БД уже неспособны справиться. Примером этого может послужить хранение изображений, звуковых файлов и видеозаписей. Существенным недостатком реляционных БД является то, что хранение реальных структур данных в виде столбцов и строк таблиц является довольно непростой задачей, поскольку при декомпозиции данных может получиться довольно большое количество таблиц, отношения между которыми сложно использовать и запомнить.

Одним из альтернативных к реляционному подходу хранения данных является объектный подход. По сравнению с реляционным подходом данный подход появился несколько позже, первые публикации, касающиеся объектно-ориентированных БД, датируются примерно серединой 1980-х годов. Объектный подход позволяет наиболее естественным способом записать объекты в БД и обеспечить их хранение и действия с ними.

1. Аспекты построения системы OLAP в объектно-ориентированных базах данных

Основой любой OLAP-системы является куб, представляющий собой упорядоченный многомерный массив и строящийся на фактических данных, которые могут находиться в нескольких таблицах фактов и обычно имеют несколько измерений [2,3].

Основными реализациями систем OLAP являются MOLAP (многомерный OLAP), ROLAP (реляционный OLAP) и HOLAP (гибридный OLAP), использующий как многомерные, так и реляционные БД. В MOLAP таблица фактов содержит все комбинации значений всех измерений и соответствующие им значения мер. В отличие от MOLAP таблица фактов в ROLAP не содержит все возможные варианты комбинаций измерений; она содержит уникальный составной ключ, который объединяет первичный ключи таблиц измерений.

Типичными схемами связи таблицы фактов с таблицами измерений являются схемы «звезда» и «снежинка». В схеме «звезда» таблица фактов является денормализированной, поскольку данная схема не позволяет учесть возможные уровни иерархии измерений. Для этого применяется схема «снежинка», главной особенностью которой является хранение информации об одном измерении в нескольких связанных таблицах. Однако главным недостатком такой схемы является усложнение в структуре базы данных, которая в данном случае будет содержать достаточно много таблиц фактов.

При сравнении реляционной модели данных с объектной моделью между ними можно найти достаточно много аналогий. Так, реляционное понятие таблицы соответствует классу, строка таблицы - экземпляру класса, столбец - свойству класса, внешний ключ - ссылке на объект и т. д. [4]. Таким образом, можно сделать вывод, что таблица фактов в объектной модели данных представляет собой класс, свойствами которого являются меры и ссылки на таблицы измерений, которые в свою очередь также являются классами.

Структуру таблицы фактов и таблиц измерений для объектной модели данных можно показать на примере базы данных некоторой торговой сети, которая имеет магазины в различных регионах страны, в которые поставляются товары различных категорий.

Факты про продажу определенного товара в определенном магазине за определенный месяц фиксируются в таблице фактов, имеющей название «Продажи». Соответственно в данной системе будет 3 измерения: «Дата», «Магазины» и «Товары». Мерой в таблице фактов является доход за продажу.

При этом каждое из измерений имеет несколько уровней иерархии. Таким образом, нижним уровнем иерархии «Дата» является «Месяц», выше находятся уровни «Квартал» и «Год». В свою очередь, для иерархий «Магазины» и «Товары» нижними уровнями являются соответственно «Магазин» и «Товар», а верхними – «Регион» (каждый магазин находится в определенном регионе) и «Категория товара» (каждый товар можно отнести к определенной категории). На рисунке 1 показана схема вышеуказанной базы данных в нотации IDEF1 [5].

В проекте стандарте объектных БД определен язык ODL (Object Definition Language – язык определения объектов), синтаксис которого аналогичен синтаксису C++ и Java, однако точно с ними не совпадает [6].

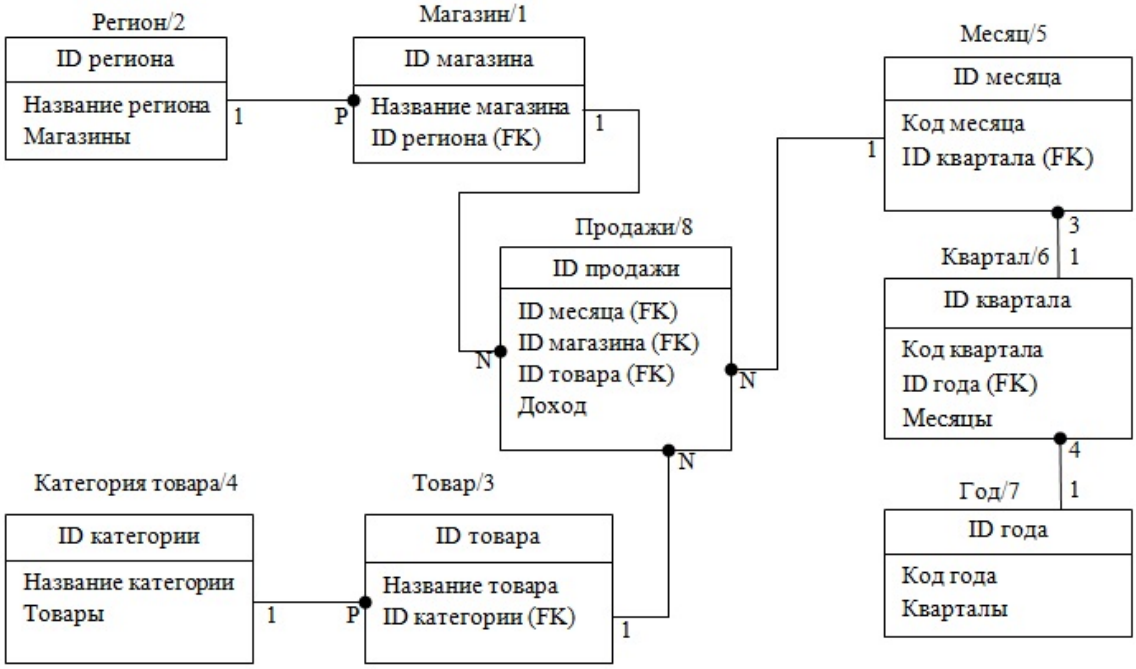


Рис. 1 – Схема базы данных торговой сети

Описание класса «Продажи» на языке ODL будет выглядеть следующим образом:

```
class FactSales
{
   attribute Month factmonth; //ссылка на объект класса «Месяц»
   attribute Shop factshop; //ссылка на объект класса «Магазин»
   attribute Goods factgoods; //ссылка на объект класса «Товар»
   attribute float profit; //доход (мера таблицы фактов)
}
```

Таким образом, в таблице фактов содержатся ссылки на экземпляры классов измерений нижнего уровня иерархии. В свою очередь, классы измерений в своем описании содержат атрибуты связи с экземплярами классов более высоких уровней иерархии. Например для класса «Месяц»:

```
class Month
{
   attribute string CodeMonth; //код месяца в БД (например, "01-2013")
   relationship Quarter quarter
     inverse Quarter::months; //описание связи с классом "Квартал" (месяц
                              //всегда принадлежит к определенному
                              //кварталу, квартал имеет 3 месяца)
}
```

В классе «Квартал» также имеется вышеуказанная связь с классом «Месяц», а также связь с классом «Год».

```
class Quarter
{
   attribute string CodeQuarter; // код месяца в БД (например, "1 кв-2013)
   relationship set<Month> months
     inverse Month::quarter; //описание связи с классом "Месяц", в данном
                             //случае атрибут представляет множество
                             //экземпляров класса "Месяц", поскольку в
                             //квартале более чем 1 месцяц
   relationship Year year
```

```
      inverse Year::quarters //описание связи с классом "Год", в котором
                             //также определена соответственная связь
}

class Year
{
   attribute integer CodeYear;
   relationship set<Quarter> quarters
     inverse Quarter::year;  //соответствующая связь с классом «Квартал»
}
```

Соответствующие связи между собой имеют классы и других измерений. В отличие от таблицы фактов, которая содержит ссылки на экземпляры классов измерений, классы измерений содержат атрибуты связи с классами измерений более высокого уровня для того, чтобы не нарушить целостность базы данных.

При агрегировании данных на более высоком уровне к измерениям можно обращаться с помощью точечного синтаксиса, применяемого в объектной модели данных, например:

Month.Quarter.CodeQuarter – код квартала, в который входит месяц;

Month.Quarter.Year.CodeYear – год, в который входит месяц (доступ через объект класса «Квартал»).

С помощью точечного синтаксиса можно также обратиться к значениям более высокого уровня иерархии к более низкому, например:

Year.Quarter[n].CodeQuarter – код n-го квартала года;

Year.Quarter[n].Month[m].CodeMonth – код m-го месяца n-го квартала года;

Таким образом, становится возможным агрегирование по измерениям с уровнями иерархий большого количества. При этом нет избыточности данных, которое возникает при ненормализированности таблицы фактов в MOLAP и схеме «звезда» в ROLAP. С другой стороны, агрегирование по более высоким уровням иерархии не будет таким затратным по времени, как при использовании схемы «снежинка» в ROLAP. Так объектная модель данных может решить две проблемы при проектировании OLAP, а именно, проблему избыточности данных, возникающую при использовании денормализированных таблицах, и проблему увеличения количества операций при нормализации базы данных, что приводит к снижению скорости работы системы в целом.

2. Использование метаданных при построении OLAP-системы в ООСУБД

Ниже рассматриваются вопросы проектирования и реализации системы OLAP для объектно-ориентированных СУБД. Данная задача является одной из задач научного исследования, касающегося интеграции информационных технологий баз данных, баз знаний и Data Mining. Так, с использованием объектной модели данных были разработаны OLAP-кубы для трёх различных баз данных: торговой компании, продажи автомобилей и компьютерной сети. Для построения кубов использовался алгоритм многопозиционного агрегирования многомерного массива [7]. Все три базы данных относятся к совершенно разным предметным областям. Поэтому возникла задача создания OLAP-системы, которую можно было бы использовать для произвольной объектной базы данных. При этом возникает одна острая проблема: OLAP-средство не может работать с произвольной объектной базой данных в связи со статическим встраиванием схемы базы данных в объектное приложение [8]. Поэтому можно сделать вывод, что для реализации технологий OLAP (а затем и Data Mining) необходимо использовать метаданные. В работе [9] предлагается описание следующих метаданных, представляющих собой метаклассы: объект, простой тип, сложный тип, атрибут, параметр, связь между классами. На их основе в данной работе предлагается добавить такие метаклассы: пакет, таблица фактов, измерение, мера, куб. Диаграмма данных метаклассов представлена на рис. 2.

Метакласс «Таблица фактов» вводится потому, что таблица фактов является специфической таблицей, в которой должны быть отображены измерения и меры для построения OLAP-куба.

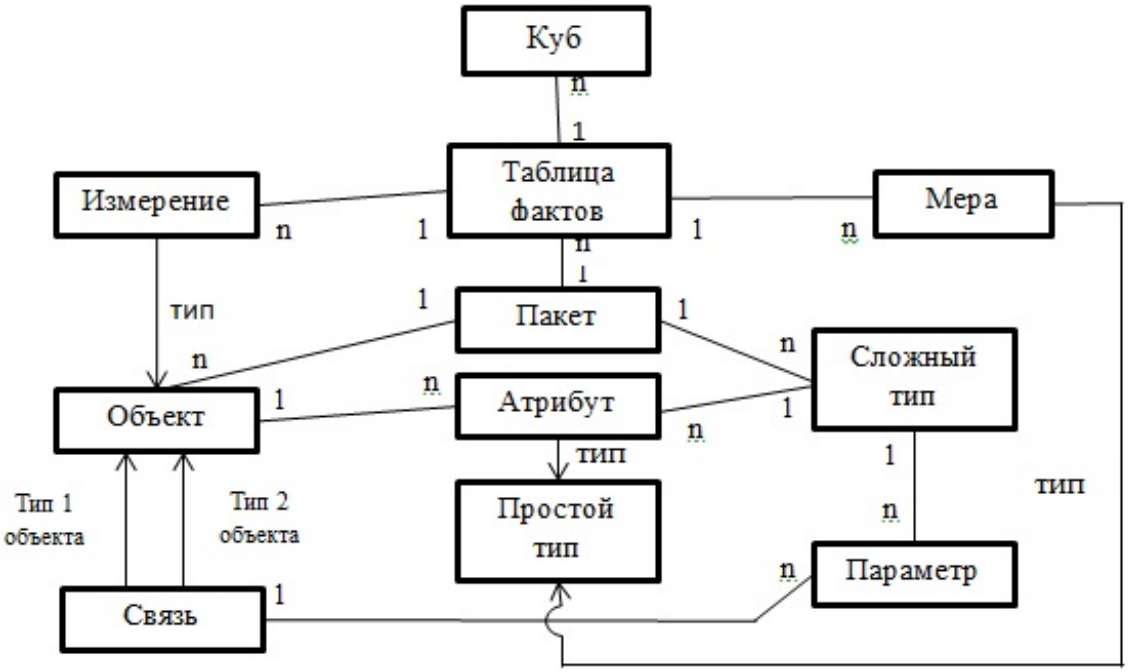


Рис. 2 – Диаграмма метаклассов OLAP-системы в объектно-ориентированной СУБД

В данном метаклассе хранится информация про название соответствующей таблицы фактов, ее измерения, меры и кубы.

```
class FactTable
{
        attribute string Name;  //название таблицы фактов
        attribute date CreationDate; //дата создания
        relationship Package ParentPackage
         inverse Package::FactTables;  //связь с метаклассом «Пакет», пакет
                                       //(схема данных) может иметь
                                       //несколько таблиц фактов
        relationship set<Dimension> Dimensions
         inverse Dimension::facttable //связь с метаклассом «Измерение»
        relationship set<Measure> Measures
         inverse Measure::facttable  // связь с метаклассом «Мера»
}
```

Метакласс «Измерение» хранит информацию о наборе доменов, по которым будет строиться многомерное пространство. Измерение относится к определенной таблице фактов, его типом будет некий предметный класс, определенный в системе. Также в данном метаклассе хранится количество значений измерения, список значений и специальный параметр шага, который используется при выполнении алгоритма многопозиционного агрегирования многомерного массива при построении OLAP-куба.

```
class Dimension
{
        attribute string Name;  //название измерения
        attribute Object Type;  //тип измерения
        attribute integer Number; //количество значений измерения (а именно
                                  //количество экземпляров класса данного
                                  //измерения)
        attribute set<String> Values; //список значения измерения
        relationship FactTable facttable
         inverse FactTable::Dimensions;
}
```

Метакласс «Мера» хранит информацию об атрибуте таблицы фактов, по которому будет происходить агрегирование. Таблица фактов может иметь несколько мер. Типом меры является некоторый простой тип (целое число, вещественное число и т. д.), определенный в

системе. Также существуют различные функции агрегирования: сумма, количество, среднее значение, минимум, максимум и т. д. Данный метакласс содержит информацию о названии меры, ее тип, функции агрегирования.

```
class Measure
{
        attribute string Name;  //название меры
        attribute SimpleType Type; //тип меры
        attribute string AggrFunc; //функция агрегирования
        relationship FactTable facttable
         inverse FactTable::Measures;
}
```

Последний из добавленных метаклассов «Куб» содержит информацию о названии OLAP-куба, дате его создания и таблице фактов, на основе которой построен данный куб.

```
class Cube
{
        attribute string Name;  //название куба
        attribute date CreationDate; //дата создания куба
        relationship FactTable facttable
          inverse FactTable::Cubes;
}
```

Выводы и дальнейшие направления исследований

На данный момент апробированы алгоритмы построения кубов OLAP для трёх баз данных с различными предметными областями [10], также реализована часть представленной на вышеуказанной диаграмме системы, представляющая подсистему проектирования базы данных посредством метаклассов и ведения данных путём создания, удаления и модификации экземпляров классов, спроектированных с помощью метаданных. Создание метаклассов позволит существенно упростить проектирование OLAP-приложений в среде объектных СКБД. Ведется также работа по реализации подсистемы OLAP в данной системе. В дальнейшем планируется проектирование подсистемы интеллектуального анализа данных (Data Mining), представляющей алгоритмы поиска ассоциативных правил в OLAP-кубах.

Литература

1. Барсегян А. А., Куприянов М. С., Степаненко В. В., Холод И. И. Методы и модели анализа данных: OLAP и Data Mining. – СПБ: БХВ-Петербург, 2004.
2. Паклин Н. Б., Орешков В. И. Бизнес-аналитика: от данных к знаниям: Учебное пособие. 2-е изд., перераб. и доп. – СПб.: Питер, 2010.
3. Харинатх С и др. Microsoft SQL Server Analysis Services 2008 и MDX для профессионалов. :Пер. с англ. – М.: ООО «И.Д. Вильямс», 2010.
4. Кирстен В., Ирингер М., Кюн М., Рериг Б. Постреляционная СУБД Cache 5. Объектно-ориентированная разработка приложений. – 3-е изд., перераб. и дополн. – М.: ООО “Бином-Пресс”, 2008.
5. Вендров А.М. CASE-технологии. Современные методы и средства проектирования информационных систем. www.citforum.ru
6. Харрингтон Д. Проектирование объектно-ориентированных баз данных: Пер. с англ. – М.: ДМК Пресс, 2001.
7. Кудрявцев Юрий, факультет ВМиК МГУ. Обзор алгоритмов MOLAP, 2008. – Режим доступа: http://www.citforum.ru/consulting/BI/molap_overview/node2.shtml.
8. Коновалов А.В. Анализ данных в объектно-ориентированных СУБД //Искусственный интеллект (Донецк), 2000, № 2, с.74-81.
9. Труб И.И. СУБД Cache: работа с объектами. – М.: Издательство ДИАЛОГ-МИФИ, 2006.
10. Фісун М.Т., Горбань Г.В. Аналіз особливостей об’єктної та багатовимірної моделей даних в СКБД Caché //Вестник Херсонского национального технического университета. – 2011. – №2(41). – С. 116-124.

УДК 004.89

ИНТЕГРАЦИЯ МНОГОАГЕНТНЫХ БАНКОВ ЗНАНИЙ С ОТКРЫТЫМИ ОБРАЗОВАТЕЛЬНЫМИ МОДУЛЬНЫМИ МУЛЬТИМЕДИА СИСТЕМАМИ

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

Для создания сетевых электронных образовательных ресурсов (ЭОР) с интерактивным мультимедиа-контентом была разработана единая программная среда функционирования – открытая образовательная модульная мультимедиа-система (ОМС) [1,2]. Интеграция сетевых ЭОР, разрабатываемых в среде ОМС, которые, благодаря своим преимуществам, получили наименование электронных образовательных ресурсов нового поколения (ЭОР НП), с многоагентными банками знаний обеспечивает дальнейшее развитие современных образовательных технологий. Благодаря этой интеграции ЭОР становятся полноценным инструментом образовательной деятельности, который может использоваться для распределения учебных заданий и мониторинга их выполнения, личностно-ориентированного обучения с выбором наиболее подходящих для пользователя учебных материалов и интерактивных образовательных модулей, а также для реализации вопросно-ответных отношений.

Совокупный контент ОМС разделен на модули, соответствующие тематическим элементам и компонентам учебного процесса. При этом каждый модуль может иметь аналог–вариатив, отличающийся элементами содержания, методикой, технологией исполнения. Электронный учебный модуль (ЭУМ) является автономным, содержательно и функционально полным образовательным ресурсом, предназначенным для решения определенной учебной задачи.

Общая архитектура ОМС объединяет две составляющие: серверную, единую для множества пользователей, и клиентскую, расположенную на рабочем месте каждого пользователя.

На сервере хранится совокупный контент ЭОР, представляющий собой структурированное множество интерактивных образовательных модулей (ИОМ). Серверная часть ОМС объединяет средства хранения и поиска требуемых ИОМ, совокупность объектно-ориентированных пользовательских интерфейсов и интернет-сервисов, удовлетворяющих дополнительные запросы пользователей, связанные с данной предметной областью. Так как серверная часть электронных образовательных ресурсов представляет собой набор хорошо известных интернет-сервисов, то в качестве хранилища совокупного контента ЭОР может выступать любой интернет-сайт или портал.

Серверная часть обеспечивает выполнение следующих функций:

- централизованное хранение ЭОР по предметным областям в виде совокупности ИОМ;
- разграничение прав доступа при получении и публикации ИОМ;
- поиск, выбор и выдача ИОМ по запросу пользователя;
- выдача выборки из метаданных указанного пользователем ИОМ.

Клиентская часть обеспечивает выполнение следующих функций:

- получение информации о доступных ЭОР и составляющих их ИОМ;
- доставка избранных ИОМ на рабочее место пользователя;
- организация локального хранилища избранных ИОМ на рабочем месте пользователя;
- воспроизведение ИОМ.

У пользователя имеется два основных варианта доступа к интерактивным образовательным модулям: с помощью поисковых запросов и посредством объектно-ориентированных интерфейсов, обеспечивающих навигацию и предметный, интуитивно ясный выход на тот или иной ИОМ.

Совокупный контент ОМС состоит из предметных ЭОР, каждый из которых, в свою очередь, является совокупностью электронных учебных модулей. Электронные учебные

модули представляют собой законченные интерактивные мультимедиа-продукты, нацеленные на решение определённой учебной задачи. Из электронных учебных модулей трех типов (модулей получения информации, практических занятий и аттестации), составляющих содержание законченного учебного материала, формируются более крупные учебные блоки (разделы, темы, курсы, предметы, дисциплины, образовательные области).

Для того, чтобы несколько отдельно взятых модулей ОМС могли объединиться друг с другом или с себе подобными и составить целостный электронный образовательный ресурс, они должны иметь стандартизованный интерфейс. Унификация архитектуры ИОМ обеспечивает соблюдение международных соглашений, упрощает модификацию модуля пользователем и позволяет автоматизировать проведение интегральной оценки качества.

Для описания ЭОР используется стандарт описания образовательных объектов Learning Object Metadata (LOM). Назначение данного стандарта – способствовать поиску, оценке, приобретению образовательных объектов и их использованию путем создания каталогов и хранилищ. Построение метаданных интерактивных образовательных модулей основано на национальной версии LOM, адаптированной к системе российского образования RUS_LOM. На основе информационной модели RUS_LOM разрабатывается профиль метаданных ИОМ. Профиль предусматривает необходимые расширения словарей RUS_LOM, а также дополнение информационной модели рядом новых элементов и ассоциируемых с ними словарей.

Метаданные являются основным инструментом систематизации информационных ресурсов в целом и ЭОР НП, в частности. Таким образом, в рассматриваемом контексте метаданные представляют собой описание ЭОР НП, реализованное через описания составляющих его электронных учебных модулей. Метаданные могут быть как частью ЭОР НП, так и храниться отдельно от него.

ЭОР НП поступают в хранилище электронных образовательных ресурсов вместе с метаданными, подготовленными их разработчиками. Метаданные могут быть представлены в виде:

- отдельного XML-файла, входящего в состав электронного учебного модуля;
- блока XML-данных, входящего в манифест дистрибутивного пакета ЭОР НП.

Особенности, вытекающие из назначения разрабатываемого информационного ресурса, отражаются в формируемом для него функциональном профиле стандартов – ссылочной модели. Примером ссылочной модели служит Sharable Content Object Reference Model (SCORM), которая определяет принципиальные технические решения для информационно-образовательной среды (ИОС), в которой реализуются процессы электронного обучения на основе Web-технологий.

SCORM базируется на концепции образовательных объектов (OO–learning objects), предусматривающей декомпозицию учебного материала на относительно небольшие единицы контента, рассчитанные на многократное применение в разных самостоятельных и независимых контекстах. Технические решения, специфицируемые SCORM, в целом относятся к уровню прикладных сервисов. Текущая версия SCORM в явном виде не привязана к сервисно-ориентированной архитектуре. В спецификации SCORM намечены направления ее дальнейшего развития, связанные как с совершенствованием представленных в ней компонентов и наращиванием их возможностей, так и с расширением состава охватываемых вопросов.

Концепция образовательных объектов в SCORM реализуется на основе обобщенной архитектуры ИОС, базовыми компонентами которой являются хранилища образовательных объектов, системы управления учебным процессом (LMS - Learning Management Systems) и клиентская среда исполнения приложений (для учащегося – это среда взаимодействия с образовательными объектами). Ключевую роль в организации электронного обучения играет LMS – серверное приложение, реализующее комплекс функций администрирования учебной деятельностью, управления контентом (выбора образовательных объектов из хранилищ и

агрегации контента), доставки его учащемуся, управления навигацией по контенту, контроля за ходом и результатами работы учащегося, формирования отчетов и др.

LMS обеспечивает поддержку планирования учебного процесса и определения заданий для учащихся, а также взаимодействия учащихся и преподавателей. Именно LMS определяет, какой контент и когда должен быть предоставлен учащемуся с учетом целей его подготовки, индивидуального задания, степени его выполнения (результатов предыдущей работы), сделанных ранее настроек интерфейса и предпочтений, зафиксированных в персональном профиле. LMS также отвечает за регистрацию и авторизацию пользователей, и обмен информацией с другими системами ИОС.

Решая проблему совместимости, стандарт SCORM не предъявляет требований к внутренней семантике знаний и основан только на предоставлении дополнительной информации в виде метаданных о целях, достигаемых при изучении текущего блока. Внутренняя семантика знаний при этом скрыта от средств автоматизированной обработки. Для обеспечения более эффективной работы системы необходимо разработать способ представления данных, обеспечивающий дополнительные возможности автоматического анализа знаний на основе смысловых связей.

В настоящее время информационные системы открытого доступа к электронным образовательным ресурсам, а также информационные системы, принадлежащие отдельным образовательным учреждениям и юридическим лицам, внедряющим современные обучающие технологии, разрабатываются в соответствии с архитектурой клиент-сервер, являющейся базовой для реализации распределенных вычислений. Дальнейшим развитием современных дистанционных образовательных технологий является интеграция ОМС с многоагентными банками знаний (МБЗ) [3, 4, 5].

Многоагентные банки знаний представляют собой распределенные интеллектуальные информационные системы учебного назначения, которые интегрируют функции интеллектуальных учебных сред (ILE, Intelligent Learning Environments) и интеллектуальных обучающих систем (ITS, Intelligent Tutoring System). МБЗ включают общие и специальные знания о предметной области, о процессе обучения и модели обучаемого, ассоциируя их с реактивными и когнитивными программными агентами, которые реализуют процедуры обработки этих знаний, формируют и выдают ответы на запросы пользователей, осуществляют адаптивное обучение. Агенты МБЗ могут взаимодействовать с ИОМ и имеют доступ к хранилищам учебных и информационно-справочных ЭОР.

Многоагентные банки знаний (МБЗ) являются результатом эволюции объектно-ориентированных методологий, технологий и инстументальных средств и систем, базирующихся на знаниях (Knowledge Based Systems – KBS). В МБЗ вместо одного интеллектуального решателя (механизма обработки знаний), используется сеть программных агентов. Многоагентный подход предполагает, что в процессе обработки знаний о предметных областях (ПО) агенты осуществляют поиск и логическую обработку знаний, взаимодействуя друг с другом посредством передачи сообщений. В процессе обработки знаний о ПО агенты формируют ответы на запросы пользователей в результате спецификации свойств сущностей ПО и вычисления каузальных, временных и других отношений на множестве этих сущностей, а также в результате планирования решения задач. МБЗ формируют ответы на запросы пользователей о значениях различных характеристик объектов и событий, о сравнении и анализе событий, выявлении связей между событиями, а также о синтезе планов действий для решения тех или иных задач (т. е. о формировании упорядоченных совокупностей событий, обеспечивающих эти решения). Необходимые вычисления, при этом производятся путем выполнения продукционных, редукционных и трансформационных правил.

Подобно архитектуре SOA (Service-Oriented Architecture), агенты реализуют систему служб (сервисов) с доступными потребителям стандартизированными интерфейсами, что уменьшает их степень связанности. Слабая связность программных агентов облегчает идентификацию естественного параллелизма, существующего в контексте предметной

области, и понимание того, как следует проводить декомпозицию работ, которые можно выполнять одновременно. Использование для формирования ответов на запросы пользователей нескольких программных агентов, параллельно реализующих вычисления, расширяет возможности МБЗ по предоставлению пользователям обобщенной информации.

Программные агенты обладают такими свойствами, как интерактивность или общественное поведение (social ability) (т.е. способны функционировать в сообществе агентов, инициируя взаимодействия и обмениваясь сообщениями с помощью некоторого языка коммуникаций); реактивность (обладают способностью воспринимать окружающую среду и своевременно реагировать на события недетеминированным образом); проактивность и целеустремленность (действуют в упреждающей манере, в частности, генерируют новые цели и действуют рационально для их достижения).

Таким образом, главным отличием программного агента от объектно-ориентированного сервера является его интеллектуальность (intelligent). С точки зрения объектно-ориентированного программирования это свойство проявляется в том, что агент содержит не только типичные для объектных классов методы и атрибуты (члены-данные класса), но также когнитивные структуры данных (CDS, Cognitive Data Structures) и методы, реализующие логические выводы, а также механизмы самообучения и адаптации. Лежащая в основе МЗБ нейрологическая модель позволяет интегрировать механизмы логического вывода и нейросетевое обучение, объединив в моделях агентов семиотический и коннекционистский подходы в искусственном интеллекте.

Другое важное свойство агентов - мобильность - позволяет динамически перераспределять вычислительную нагрузку в зависимости от состояния сети, а также обеспечивает интероперабельность между различными существующими и разрабатываемыми системами. Интероперабельность достигается при стандартизации таких аспектов, как передача агентов и служебных (используемых агентом) классов между агентными системами, а также управление агентами.

Большинство распределенных систем учебного назначения реализует интероперабельность за счет открытости интерфейсов доступа к своим сервисам и/или путем использования единого формата для обмена данными, а именно - расширяемого языка разметки XML (eXtensible Markup Language) и связанной с ним объектной модели представления документов DOM (Document Object Model), осуществляя, если необходимо, XSL-преобразования. Такой подход позволяет решить задачу синтаксической интероперабельности. Однако, для организации взаимодействия между различными образовательными системами в сети Internet в большинстве случаев недостаточно обеспечения только синтаксической интероперабельности. Это обусловлено прежде всего тем, что одну и ту же информацию можно синтаксически по-разному представить, и, как следствие, может возникнуть естественный барьер между системами.

Проблема отсутствия четких семантических определений мешает объединению образовательных систем различных производителей. Решение проблемы состоит в построении семантической сети с использованием языка сетевых онтологий OWL (Ontology Web Language), которая дает возможность программным агентам понимать семантику документов и данных. OWL может использоваться, чтобы явно представлять значения терминов и отношения между этими терминами в словарях. Такое представление терминов и их взаимоотношений называют онтологией, под которой в настоящее время понимается любое описание декларативных знаний, сделанное на формальном языке и снабженное некоторой классификацией специфицируемых знаний, позволяющей удобно их воспринимать. Язык сетевых онтологий OWL имеет больше средств для выражения значения и семантики, чем XML и система описания ресурсов RDF (Resource Description Framework), и, следовательно, способен лучше представить поддающийся машинной обработке сетевой контент. Язык OWL позволяет не только описывать классы и свойства, но также задавать ограничения на их использование. На языке дескрипционной логики это означает, что логика, лежащая в

основе OWL, содержит кроме описания отношений также и аксиомы, задающие соотношения между данными отношениями и различного рода ограничения последних.

В рамках учебных процессов применение Web-онтологий позволяет специфицировать основные компоненты учебных дисциплин - лекции, практические занятия, лабораторные работы, используемые учебные материалы, а также обеспечивает возможность организации эффективного доступа к распределенным учебным ресурсам, путем создания многоагентной базы знаний, в которой интеллектуальные программные агенты реализуют запросы пользователей, обобщают информацию и строят индивидуальные образовательные траектории, выбирая наиболее подходящие для пользователя учебные материалы, интерактивные образовательные модули и тестовые задания.

Учебные электронные образовательные ресурсы представляют собой электронные учебные пособия, содержащие систематизированный материал в рамках учебной программы. Они реализуют полный цикл по преподаванию учебной дисциплины (или ее раздела), то есть обеспечивают предоставление учебного материала, отработку навыков, контроль знаний. Такого рода ЭОР призваны расширить возможности преподавателя в выборе и реализации методик обучения и в то же время предоставить ученикам возможности для самостоятельной работы. Информационно-справочные ЭОР обеспечивают информационную поддержку учебного процесса. Такие ЭОР нацелены на использование в качестве источников данных и знаний при решении творческих учебных задач, в том числе выходящих за рамки учебных программ.

Использование многоагентных банков знаний обеспечивает эффективное развитие и неограниченное расширение открытой образовательной модульной мультимедиа системы по мере получения новых знаний о предметной области, создания новых, эффективных методик представления образовательной информации, практических заданий, форм и методов контроля полученных знаний, умений и навыков. Интеллектуальные агенты МБЗ могут использоваться в распределенной системе открытого доступа к электронным образовательным ресурсам для распределения учебных заданий и мониторинга их выполнения, для формирования индивидуальных образовательных траекторий и личностно-ориентированного обучения с выбором наиболее подходящих для пользователя учебных материалов и интерактивных образовательных модулей, а также для реализации вопросно-ответных отношений.

Литература

1. Осин А.В. Электронные образовательные ресурсы нового поколения: открытые образовательные модульные мультимедиа системы.// Интернет-порталы: содержание и технологии. Сб. науч. ст. Вып.4. – М.: Просвещение, 2007.
2. Осин А.В. Открытые образовательные модульные мультимедиа системы. − М.: Издательский сервис, 2010.
3. Зайцев Е.И. Об агентно-ориентированных системах и многоагентных банках знаний. // Материалы VI Международной научно-практической конференции “Объектные системы - 2012”. – Ростов-на-Дону, 2012.
4. Зайцев Е.И. О концепции многоагентных банков знаний, как интеллектуальных обучающих системах // Материалы 16-й международной конференции “Современное образование: содержание, технологии, качество”.– СПб.:Изд-во СПбГЭТУ “ЛЭТИ”. 2010.
5. Миронов А.С., Зайцев Е.И. О концепции многоагентных учебных сред, называемых многоагентными статическими банками знаний. // Материалы XVII Международной конференции “Современное образование: содержание, технологии, качество”.– СПб.:Изд-во СПбГЭТУ “ЛЭТИ”. 2011.
6. Описание OWL. [Электронный ресурс]. – Режим доступа: http://www.w3.org/TR/owl-ref/.
7. Проект Semantic Web. [Электронный ресурс]. – Режим доступа: http://www.w3c.org/sw
8. Спецификация языка RDFS. [Электронный ресурс]. – Режим доступа: http://www.w3c.org/rdfs
9. Метаданные для информационных ресурсов сферы образования. Спецификация информационной модели [Электронный ресурс]. – Режим доступа: http://spec.edu.ru.

УДК 004.04

РЕАЛИЗАЦИЯ ОДНОПОТОЧНОГО КОММУНИКАЦИОННОГО ПО

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

Введение

При модернизации распределенной системы обработки данных АСУТП АЭС была поставлена задача замены коммуникационного ПО. Рассматривались действующее ПО на основе CORBA [1], реализующее клиент-серверное взаимодействие, технология DDS [2] на основе модели публикация-подписка и новая разработка ПО для этих задач. При этом нужно было повысить качество сетевого взаимодействия.

Действующая система была реализована в виде многопоточного пользовательского ПО с блокировками потоков до получения данных. При сбоях связи и зависаниях сети система штатно переходила на резервированные сети, с выдерживанием задержек, но время от времени приходилось сталкиваться с присущими CORBA проблемами [3]: "ошибками, относящимися к надежности потоков управления, надежности исключительных ситуаций и управлению памятью". Опыт эксплуатации показывал не всегда правильное функционирование при сложных условиях связи, ошибках в сети и переходах на резервные маршруты передачи данных. Кроме того, архитектурно сложно выглядели системы, требующие модели публикации-подписки.

Второй подход, в свою очередь, не слишком хорошо решает не присущие DDS клиент-серверные задачи. Предлагаемый [2] протокол RTPS использует UDP как транспортный уровень и переносит механизмы гарантированной доставки на пользовательский уровень. Это не первый и не последний способ реализации функциональности TCP в пространстве пользователя. Является ли это решение не худшим, чем транспортная реализация TCP, - это вопрос, требующий тщательного рассмотрения.

С точки зрения приложения важен еще момент буферизации данных, которые получены по UDP. DDS предоставляет многопоточные сетевые сервисы для этих целей, которые добавляют еще один уровень сложности.

Учитывая вышесказанное, было принято решение разработки коммуникационного ПО TA (англ. Transparent Architecture), призванного совместить клиент-серверную модель и модель публикации-подписки. Технология Transparent Architecture построена на основном временном цикле одного потока, который осуществляет прием и передачу данных без единой блокировки. Это, кстати, дает дополнительное преимущество, т.к. "все переключения контекста при получении каждого пакета сильно расходуют время центрального процессора и существенно снижают производительность сети" [4] .

1. Основные принципы реализации

Неблокируемость и немодальность – ключевые особенности Transparent Architecture. Клиентские программы в клиент-серверных приложениях часто построены по принципу выделения отдельного потока для приема данных от сервера. Типичными особенностями такой схемы является модальность, заключающаяся в переходе в режим ожидание чтения данных с таймаутом. Поток при этом блокируется, при обрыве связи происходят переходы от одного таймаута к другому.

Транспортный уровень построен на неблокирующих способах взаимодействия (в частности, сокетах). Прием данных осуществляется при наличии данных без ожидания, передача – без ожидания завершения, которое будет зафиксировано позднее. Главный цикл обеспечивает точность поддержания времени в соответствии с требованиями задачи и возможностями операционной системы.

Для Оберон-версии управление транспортным уровнем задается абстрактным типом Channel.

```
Channel* = POINTER TO ABSTRACT RECORD END;
```

Конкретные реализации TCPChannel, UDPChannel, MUDPChannel, MSHMChannel, ICMPChannel подключаются в зависимости от конфигурации с помощью паттерна "Абстрактная фабрика" [5]. Преимуществом объектно-ориентированного подхода является то, что любой из существующих протоколов или вновь разработанный может быть введен независимо от базовой части системы.

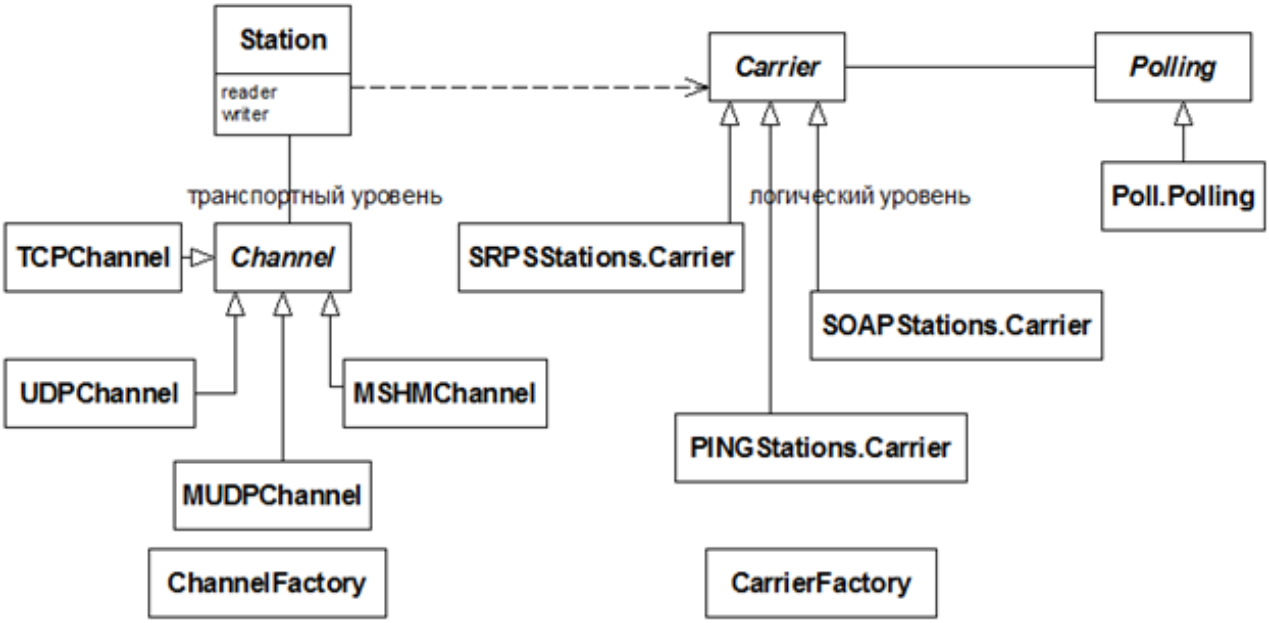


Рис. 1 – Иерархии типов транспортного и логического уровней

Transparent Architecture предлагает гибридный подход, меняя для разных задач транспортный уровень протокола:

- TCP – для клиент-серверных приложений;
- multicast UDP (MUDP) - для приложений публикации-подписки.

Аналогично задается логический уровень, т.е. формат данных протокола, при этом реализация SRPSStations формирует данные бинарного формата, SOAPStations формирует данные XML формата, а PINGStations формирует структуру ICMP пакета.

Объект типа Station играет роль паттерна "мост" в терминологии [5], позволяя разделить иерархии классов Channel и Carrier. Таким образом, транспортный и логический уровни протокола обмена могут развиваться и использоваться независимо друг от друга.

Паттерн стратегия из [5] используется для задания алгоритмов поллинга. Это сделано для оптимизации быстродействия системы в будущем: планируется ставить более эффективные стратегии, как epoll для Linux и Connection Points для Windows. Преимущество ОО-подхода здесь - это гибкость вплоть до управления базовыми алгоритмами.

2. Организация уровня данных

При проектировании было решено отказаться от блокирующих вызовов удаленных функций через прокси. Типичное решение для CORBA-клиентов вызов

```
Fun(x_in, y_out);
```

превращается в пару запрос-ответ. При этом единицей обмена становится не набор параметров функций, а сами параметры, представляющие собой структуры любой сложности.

Обмен данными происходит через специальные объекты t: Table, закрепленные за каждым типом данных. Это позволило использовать механизмы, при котором создаются цепочки программ, выполняющиеся в строгом порядке по мере возникновения некоторых сетевых событий. Для Клиент-Серверных приложений примером может служить цепочка:

```
v.val := "grep -i -r byte_order /usr/include/*";
```

```
t.Request(v, TRUE, ProcessResults, NIL);
t.Call(NextRoutine, NIL);
```

которая создает запрос поиска файлов по шаблону, вызывает его на удаленном сервере, по событию – ответу на запрос – читает данные в переменную stringsList и вызывает программу обработки результата ProcessResults, после чего управление переходит к следующей программе NextRoutine. Все операции выполняются в строгой последовательности без блокировки основного потока. Ошибки разрывают цепочку выполнения, вызывая соответствующие обработчики.
Для обмена по подписке примерная цепочка такая:

```
v.val := "(value < 23.3) OR (value > 76.3)";
t.Write(v, TRUE);
t.Subscribe(TRUE, OnNextValue, NIL);
```

она уже выполняет многократно следующую за ней операцию OnNextValue при каждом ответе от удаленного узла, до тех пор, пока не сменится данная подписка, т.е. до вызова следующего t.Write.

Также было принято решение отказаться от IDL-компилятора, а использовать RTTI-информацию в тех реализациях, где она имеется. Для языков программирования, дающих доступ к информации времени выполнения (RTTI – RunTime Type Information), таких как BlackBox Оберон, используется RTTI. Для языков, не дающих подобного доступа, как C, используются специальные макросы, извлекающие эту информацию из текста определений переменных. В реализации Оберон-2 XDS информация, предоставляемая системой, является неполной, ее пришлось также расширять.

Протокол SRPS рассчитан на передачу данных сложных типов, включая составные структуры, массивы и динамические массивы, реализуемые через указатели в языках программирования. Динамические массивы могут содержать структуры, элементы которых являются динамическими массивами.

Информация о типах служит для сериализации пользовательских данных в сетевые посылки. При этом передаваемые по сети данные:

- Упакованы и распаковываются в структуры с учетом выравнивания на машине клиента;
- Все числа имеют сетевой порядок байт (big-endian);
- Динамические массивы передаются с длинами и размеров элементов;
- Каждый передаваемый тип данных сопровождается двумя контрольными суммами, и на стороне клиента данные эти могут отклоняться при несоответствии контрольных сумм.

3. Управление сервисом

В протоколе SRPS каждому уровню (Station, Channel, Table) соответствует свой набор параметров Качества Обслуживания (QoS – Quality of Service), с помощью которых узлы опеделяются по временным и надёжностным характеристикам информационного потока. Эти параметры устанавливают таймауты, размеры буферов обмена и параметры систем связи.

Временные характеристики задают период главного цикла, период обмена конкретной станции, интервал проверки работоспособности (пингования) станции, таймаут пингования, интервал установления соединения (для протоколов с соединением, как TCP), интервал ошибки таймаута для определенных запросов. Конфигурационные характеристики задают имена хостов и IP-адреса, номера портов.

Изменение конфигурации может проводиться в режиме онлайн, путем рассылки сообщений, содержащих структуру QoS и номер абонента. Получатель обрабатывает сообщение стандартным обработчиком HandleMsg и меняет параметры соединения.

4. Макетирование и Реализация приложения

При макетировании приложения распределенной обработки данных АСУТП, требующего и клиент-серверных операций, и публикации-подписки были оценены варианты реализации на DDS и на ТА. В качестве DDS сравнивалась OpenSplice, community edition, версия 5.2, платформа Linux 2.6 для X86. В DDS-реализации потребовался промежуточный сетевой сервис networking, с которым работает приложение через общую память на локальной машине. На сервисе была включена конфигурация BestEffort, конфигурация reliable с обеспечением механизма последовательности и гарантии передачи данных была отключена. Следующая тестовая схема была реализована. Три программных узла обмениваются короткими посылками на 3-х разных компьютерах (10.0.201.216 10.0.201.206 10.0.28.5) с интервалом 20 мс. Из-за такого явления, как «джиттер» [4], посылки приходят на адресаты с различными задержками. Буферизация по времени отсутствует, если принятая посылка не попадает в интервал, то она отбрасывается. Процент не попавших в интервал фиксируется и служит оценкой качества работы коммуникационных схем в реальном времени.

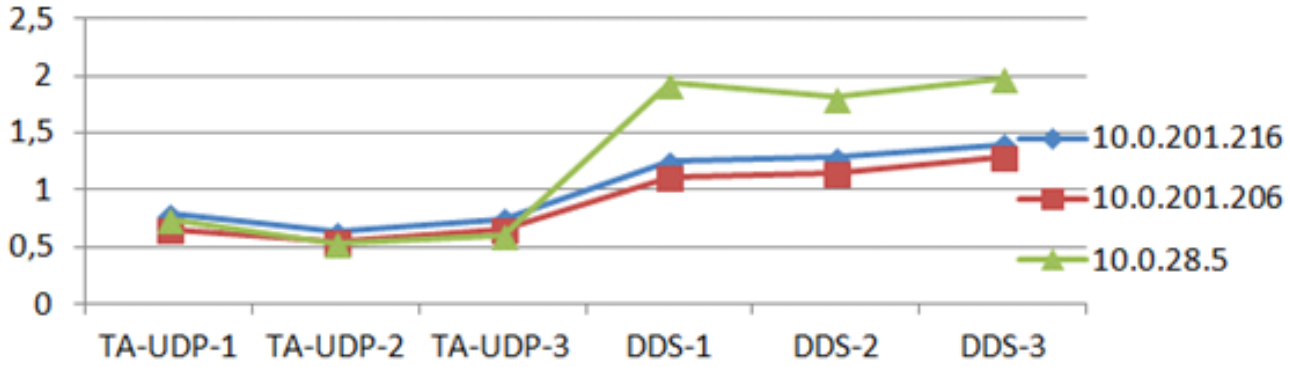


Рис. 2 – Процент числа пакетов, вышедших за границы интервала

Полученный результат свидетельствует о том, что дополнительные сервисы и потоки еще и вносят дополнительные задержки и девиации циклов, что может заметно ухудшить качество передачи данных. Естественно, буферизация на стороне клиента улучшает ситуацию, но не во всех случаях это возможно.

В результате решение было принято в пользу ТА, как гораздо менее избыточного подхода, при использовании разных протоколы для подходящих им задач. В настоящее время на ТА ведутся работы по модернизации ПО.

Первоначально коммуникационное ПО было реализовано в виде С-библиотеки с возможностью присоединения к BlackBox. Начиная с версии 2.0, появились Оберон-реализации для BlackBox и XDS. Все реализации тестировались для ОС Windows и Linux.

ТА реализована, версия 2.0 доступна в исходных текстах как ta1 на sourceforge.net.

Литература

1. Michi Henning, Steve Vinoski. Advanced CORBA Programming with C++. Addison Wesley Longman, 1999.
2. Data Distribution Service for Real-Time Systems. Version 1.2. OMG available specification. Formal 07/01/01.
3. Michi Henning. The Rise and Fall of CORBA, ACM Queue, Volume 4, Number 5, June 2006. Пер. Сергей Кузнецов.
4. Таненбаум Э., Уэзеролл Д. Компьютерные сети, 5-е изд. СПб.: Питер, 2012.
5. Э. Гамма, Р.Хелм, Р.Джонсон, Дж. Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2001.

УДК 681.3

РЕДАКТОР ОНТОЛОГИЙ ДЛЯ ОНТОЛОГОУПРАВЛЯЕМОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Грегер Сергей Эдуардович, доцент, Уральский федеральный университет имени первого Президента России Б.Н.Ельцина, Нижнетагильский технологический институт (фил.), Россия. Нижний Тагил, segreger@gmail.com
Ахметов Даниил Равилевич, студент, Уральский федеральный университет имени первого Президента России Б.Н.Ельцина, Нижнетагильский технологический институт (фил.), Россия. Нижний Тагил, dan1@liveinternet.ru

На VI конференции “Объектные системы – 2012” нами был представлен доклад на тему: “Редактор метамодели онтологической системы” [1] в котором рассматривалась возможность построения корпоративных порталов на основе CMS Plone с использованием онтологических моделей. Главной особенностью CMS Plone является использование в качестве хранилища контента объектной базы Zope Object Database (ZODB). Таким образом, целью разработки являлось создание редактора онтологий, хранимой в объектной базе ZODB. Перед нами стояла задача создать редактор, управляющий графом объектов и отображающий структуру графа в иерархическом виде.

Для хранения онтологии в объектной базе данных был разработан набор компонент, обеспечивающих представление концептов языка OWL, таких как *Class*, *Data Property*, *Object Property* и их атрибутов [2]. Для хранения индивидов онтологии был использован компонент *Individual*. Использование таких компонент позволило эффективно хранить онтологию и реализовать выполнение запросов к онтологии. При создании и модификации элементов онтологии сервер запросов манипулирует объектами указанных классов, используя средства управления объектной базы данных ZODB.

К разрабатываемому приложению были предъявлены следующие требования:

1. Редактор должен обладать возможностью загружать и визуализировать любую, представленную на языке OWL, онтологию.
2. Редактор должен предоставлять пользователю дружественный интерфейс, позволяющий быстро создавать и редактировать концепты онтологии.
3. Редактор должен предоставлять гибкий поисковый механизм, позволяющий легко искать и редактировать концепты онтологии.

Однако дальнейшие исследования привели к тому, что в разработанном редакторе был выявлен ряд недочетов касательно пользовательского интерфейса и возможности совместной разработки.

Перечислим основные недочеты предшествующей разработки:

1. Элементы онтологии представлены в виде каскада раскрывающихся списков (дерева), однако такой подход не позволяет увидеть связи между несколькими элементами онтологии.
2. При использовании онтологий больших размеров, редактор показывает очень низкую производительность, связанную с тем, что загружает все данные об онтологии при первом же обращении. Помимо нецелесообразного расходования оперативной памяти, происходит генерация очень большого количества html-разметки, что негативно сказывается на производительности браузера.
3. При изменении или добавлении новых элементов в онтологию происходит полная перезагрузка страницы, после чего данные об онтологии вновь загружаются в оперативную память, и генерируется html-разметка. В процессе моделирования, скорость заполнения онтологии пользователем довольно низка.
4. Не предусмотрены возможности совместной работы над онтологией.

Рассмотрим возможные пути устранения этих недочетов:

1. Для устранения первого недочета было решено добавить несколько дополнительных видов отображения онтологии. Так как онтология представляет собой граф, логично

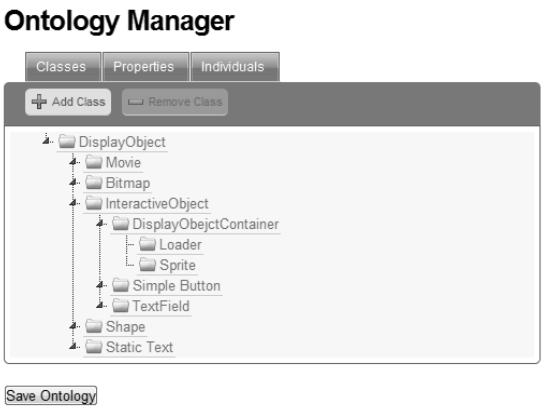
было предложить возможность визуализации элементов онтологии в виде графа (рис. 1):

Рис. 1 – Вид отображения онтологии

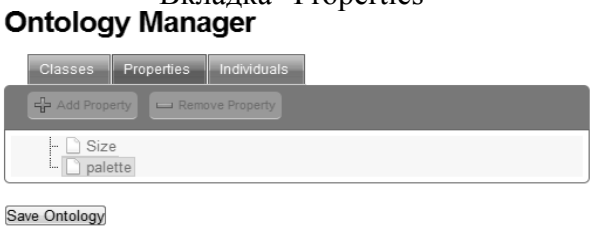
Попутно, существующее представление в виде дерева было переработано и разделено на 2 подвида (разделенный и обобщенный):

- **Разделенный вид.** Представляет собой 3 раскрывающихся дерева (на основе вложенных списков), расположенных во вкладках («Classes», «Properties», “Individuals”). Каждое дерево отображает только элементы определенного концептуала.

Вкладка “Classes”



Вкладка “Properties”



Вкладка “Individuals”

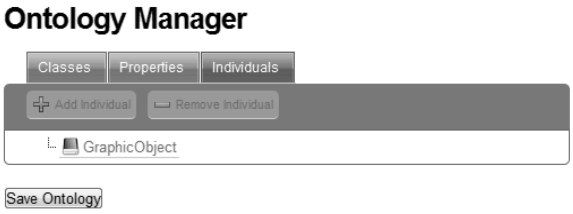


Рис. 2 – Раздельный вид

В целом, виды различных вкладок в редакторе отличаются лишь иконками в дереве элементов, а также разными названиями пунктов меню. С точки зрения реализации различий также практически нет – отличаются лишь типы представляемых концептуалов онтологии.

- **Обобщённый вид.** Совмещает в одном дереве все концептуалы и представлен на рисунке.

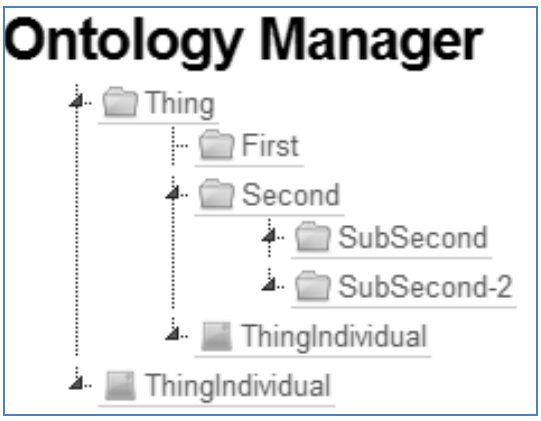


Рис. 3 – Обобщённый вид

2. Для разрешения второго недочета было решено ещё более активно использовать технологию AJAX. Поскольку любая онтология имеет иерархическую структуру, то при первом обращении к странице представления онтологии можно отображать только родительские элементы, а при развертывании одного из родительских элементов производится асинхронный запрос на сервер. После чего сервер отправляет клиенту информацию о потомках родительского элемента в формате JSON — это в значительной степени повышает производительность, как сервера, так и клиента.
3. Для решения проблемы динамического добавления и изменения элементов онтологии также использовалась технология AJAX. Предлагается 2 режима разработки онтологии: “Online” и “Offline”. Режим “Offline” предполагает работу над онтологией в оперативной памяти клиента. После окончания исправлений пользователь при нажатии на кнопку “Save Ontology” отправляет запрос с информацией обо всех измененных узлах на сервер, а тот в свою очередь производит соответствующие изменения в базе данных. Режим “Online” подразумевает, что при изменении, добавлении или удалении узла онтологии в оперативной памяти клиента производится изменение информации о текущем узле, а также на сервер отправляется соответствующий запрос на изменение информации об узле уже в объектной базе сервера. Вследствие проделанных действий в оперативной памяти клиента изменяется только лишь определенный узел, и перестраивать иерархическую структуру онтологии заново не приходится.
4. Для получения возможности совместной разработки онтологии пригоден только описанный выше режим «Online», поскольку в режиме «Offline» невозможно отслеживание состояния редактирования, вследствие чего возникнут ситуации перезаписи данных и прочие неприятные случайности. Ситуация с режимом «Online» обстоит лучше, однако необходимо постоянно отслеживать состояние редактирования и если один из пользователей пытается получить доступ к редактируемому элементу, то необходимо временно заблокировать доступ к данному объекту.

По итогам проделанной работы, можно сделать вывод, что недочёты, выявленные в предыдущей версии редактора онтологий, исправлены. Мы имеем действующее приложение для редактирования онтологий в онтологоуправляемой системе с более удобным пользовательским интерфейсом.

Литература

1. Грегер С.Э. Редактор метамодели онтологической системы // Объектные системы – 2012: материалы VI Международной научно- практической конференции (Ростов-на-Дону, 10-12 мая 2012 г.) / Под общ. ред. П.П. Олейника. — Ростов-на-Дону: ШИ ЮРГТУ (НПИ), 2012. — с. 88-92.
2. Сковородин Е.Ю. Грегер С.Э. Построение онтологического портала с использованием объектной базы // Объектные системы - 2010: Материалы I Международной научно-практической конференции. — Ростов-на-Дону, 2010 г. — с. 74-78.

УДК 004.651.54/004.652

СИСТЕМЫ МАТРИЧНЫХ УНИВЕРСАЛЬНЫХ ОБЪЕКТНО-РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

Микляев Иван Александрович, канд. физ.-мат. наук, доцент, Филиал Северного (Арктического) федерального университета в Северодвинске “СевмашВТУЗ”, Россия, Северодвинск, Ivanmia1@rambler.ru

Введение

При рассмотрении условий работы в системе МЧС на первый план выходят условия неопределённости основного места источника информации, т.к. место происшествия нельзя определить заранее. Т.к. каждое происшествие особенно по-своему, то неопределённость структуры информации налицо. С другой стороны, предъявляются максимальные требования по скорости и полноте передаваемой информации для формирования управленческого решения, поэтому требуется максимально быстро и максимально подробно получить информацию с места происшествия.

Всё это требует наличия простой локальной информационной системы на месте происшествия, которая может быстро принять любую форму информации, не упуская малейшие нюансы, независимо от наличия и формы связи с центром. Сотрудник, вносящий информацию, должен быть минимально загружен знанием системы, при этом иметь возможность максимально подробно предоставить информацию в центр.

В центре, куда стекается информация, она должна быть максимально структурирована и нормализована, что является необходимым требованием автоматизированной обработки.

Такие задачи достаточно актуальны в наше время и в военных структурах, пожарных, скорой помощи и даже лесном хозяйстве, где лесник достаточно просто может иметь ноутбук при обходе вверенной территории, при этом необходимо не зависеть от того, есть связь с центром или её нет.

Очевидным решением в данной ситуации является использование объектно-ориентированных информационных систем, с динамическим расширением структуры.

Примеры систем, предназначенных для решения подобных задач, представлены на основе матричной универсальной объектно-реляционной базы данных [1].

1. Универсальное приложение для работы с МУОРБД

Универсальное приложение представляет собой один исполняемый файл размером в несколько мегабайт, который не требует какой либо предустановки и может быть скопирован на любой носитель информации в любом месте.

После запуска основной программы развёртывается МУОРБД со всем своим инструментарием, которая по умолчанию располагается в директории рядом с программой, либо в месте указанном пользователем.

Пользователь может приступать к работе в МУОРБД сразу после запуска программы, без какой-либо дополнительной подготовки или каких-либо дополнительных настроек.

Универсальное приложение для работы с МУОРБД представляет собой всего две формы: форма администрирования МУОРБД и рабочая форма [1].

Форма администрирования МУОРБД предназначена для управления метаданными.

Рабочая форма универсального приложения МУОРБД таблиц предназначена для работы с основными данными МУОРБД.

2. Универсальный сервер универсального приложения МУОРБД

Универсальное приложение переходит в **режим сервера** выбором соответствующего пункта меню, после чего оно готово отвечать на запросы клиентских приложений.

Система взаимодействия сервера и клиентского приложения определяется внешним файлом параметров у клиентского приложения, в котором находится лишь местонахождение

сервера, а у сервера файл параметров определяет систему связи и её параметры, которые могут быть изменены даже во время работы сервера.

Форма связи может быть многоканальной, от простой передачи запроса и ответа через файл, до прямой передачи на IP-адрес через выделенный порт, с различным протоколированием TCP/IP, UDP и др.

Передача данных через порт на IP-адрес достаточно стандартная.

Использование же файлов для общения сервера и клиентского приложения - достаточно редкий случай и, на первый взгляд, очень не эффективный. Однако при разработке информационной системы очень полезно иметь максимально открытую систему контроля, также это очень эффективно при широком вещании сервера на все клиентские приложения, когда сервер загружен только при формировании ответа на запрос и образовании файла, а клиентские приложения считывают информацию самостоятельно, тогда как при сокетной технологии общения необходимо каждому клиенту направить ответ.

3. Специализированный сервер над универсальным приложением МУОРБД

При разработке МУОРБД закладывалась концепция открытого программного обеспечения.

Простота формирования МУОРБД даёт возможность среднему по квалификации программисту создать за достаточно малый срок собственную СУБД такого формата, масштабы времени сопоставимы с временем создания собственной информационной системы на базе стандартных СУБД. Таким образом, появляется новая методология создания информационных систем с нуля, которая в результате даёт возможность создавать максимально специализированную информационную систему под конкретную предметную область, с максимально эффективными производительностью и использованием аппаратных средств.

Ещё одна возможность создать полностью специализированную информационную систему состоит в том, что есть возможность создавать специализированную надстройку над универсальным приложением, просто включая его модули в свой проект. Тогда весь инструментарий универсального приложения МУОРБД полностью доступен разработчику.

Такой путь решения позволяет создать и собственный специализированный сервер над универсальным приложением МУОРБД. В этом случае важными достоинствами являются:

- возможность распределять задачи между клиентским приложением и сервером, максимально эффективно для конкретной задачи;
- использовать самые различные пути связи между ними, исходя из конфигурации аппаратных средств у пользователя и задач, которые на них возложены;
- есть возможность само серверное приложение использовать как одно из клиентских приложений;
- открывается возможность серверному приложению управлять клиентскими приложениями по требованиям поставленной задачи;
- и т.д., и т.п.

Примером можно привести задачу проведения конференции с нуля. Т.е. выбирается место проведения, например, на природе, устанавливаются несколько компьютеров, соединяются в сеть любой конфигурации. На один из них копируется серверное приложение с базой данных конференции, на остальные - клиентское приложение, или делается его общий ресурс. Указывается клиентским приложениям местонахождение сервера и параметры связи. Всё, система проведения конференции готова.

Серверное приложение управляет клиентскими приложениями по ходу конференции, пользователи клиентских приложений имеют возможность самостоятельно пользоваться клиентским приложением.

Подобная задача решалась и для проведения защиты дипломов в СевмашВТУЗе: серверное приложение было установлено у секретаря, а клиентские приложения у членов ГАК, которые под управлением секретаря автоматически получали всю информацию о

выступающем дипломнике и материалы диплома, а член ГАК мог работать с данными других дипломников или получать другую информацию по процедуре проведения защиты дипломов.

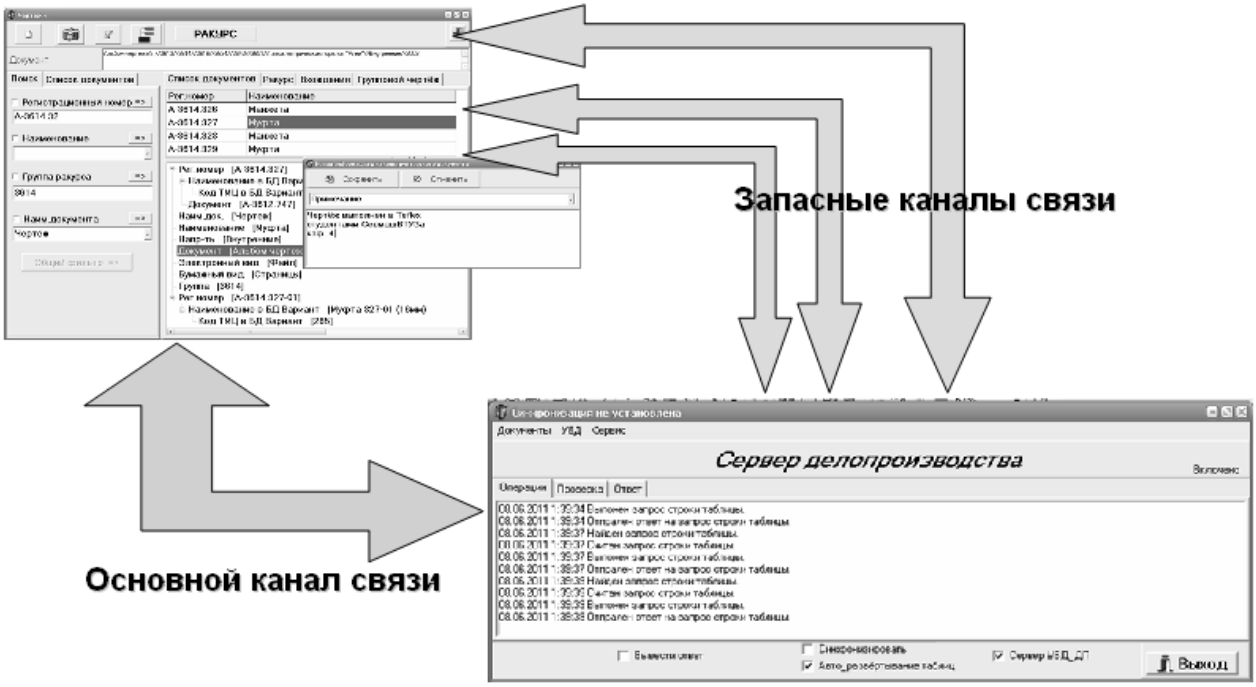


Рис.1 - Специализированный сервер над универсальным приложением МУОРБД

4. Сеть самостоятельных серверов

Сохраняют актуальность в настоящее время задачи распределённых баз данных, которые работают в единой системе и используют информацию друг друга, но при этом сохраняя возможность работать автономно при отсутствии окружения.

Например, в ВУЗе есть отдел кадров, где находятся все данные по сотрудникам и преподавателям, деканаты, где находятся данные по группам, рабочие и учебные планы, отдел кадров студентов с данными о конкретно студентах, кафедры, где происходит закрепление дисциплин за преподавателем, учебно-методический отдел, где формируется расписание, бухгалтерия, остальные отделы пока опустим.

Очевидно, что все эти подразделения тесно переплетены. По предметной области какие-то отделы могут ставить условия иметь возможность выполнять задачи автономно, например, заведующий кафедрой возьмет распределение нагрузки домой, или кто-то производит работу удалённо, могут быть условия и информационной безопасности, важным условием задачи может быть независимость от состояния сетевых ресурсов и т.д.

В системе МУОРБД предусмотрена возможность создания комплекса баз данных, с основной локальной и внешними подключаемыми [2]. В основной базе данных при подключении внешней БД появляется возможность в её таблицах делать ссылки на записи таблиц из внешних БД.

Для этого регистрируется в основной БД внешняя БД и вносится отражение необходимой таблицы в основную БД, которое состоит из ссылки в метаданных рис. 2.

5. Сеть тяжёлых дублирующихся самостоятельных серверов с лёгкими серверами-«сателитами»

Комплексы БД могут создаваться не только по структурному признаку, а и по функциональному. При решении задачи централизованного расчёта заработной платы в образовательных учреждениях, необходимо из ста учреждений организовать ежемесячный сбор табелей. Соответственно, имеются стационарные БД в образовательных учреждениях и общая БД в Управлении образования, по системе подключения внешних БД к основной были

созданы БД-сателиты табелей, которые могли сосуществовать и в образовательных учреждениях и в Управлении образованием (рис. 3).

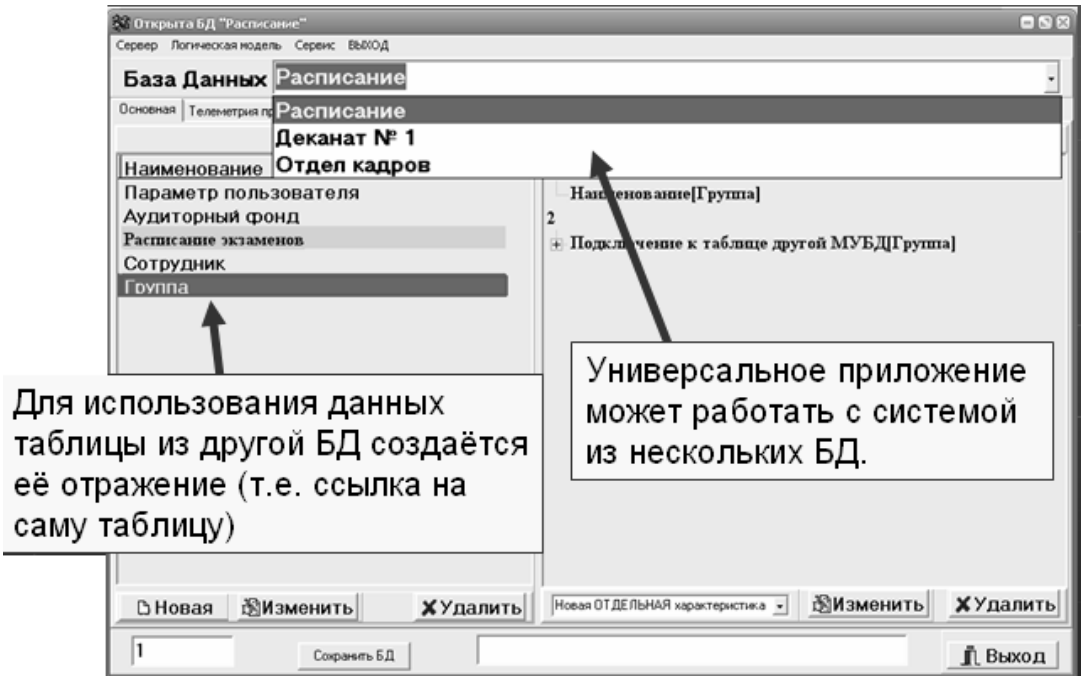


Рис. 2 - Система подключения внешней МУОРБД к основной

Таким образом, необходимо было передать по любым каналам связи или даже на дискете только эту БД-сателит.

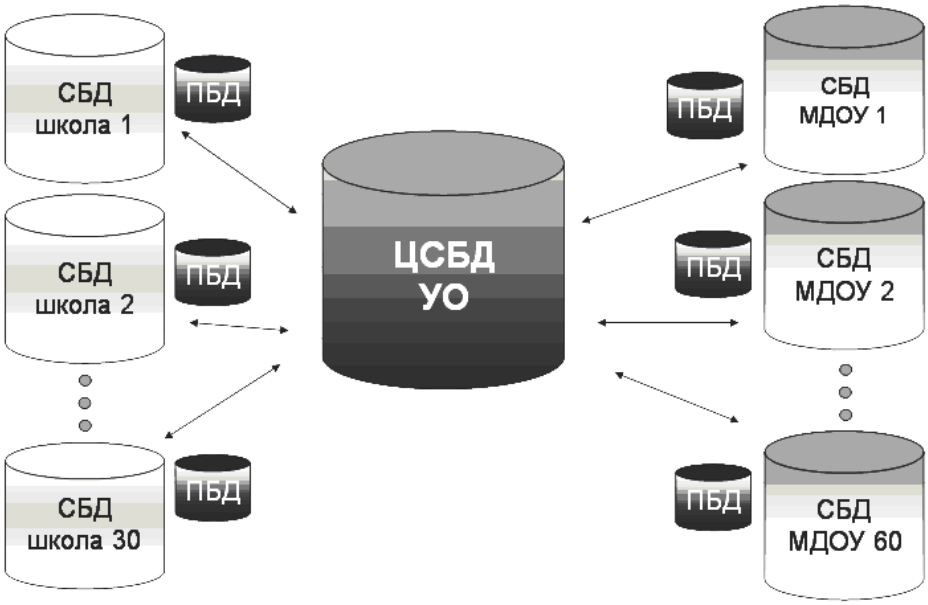


Рис. 3 - Сеть тяжёлых дублирующихся самостоятельных серверов с лёгкими серверами «сателитами»

Выводы

В данной работе показана лишь малая часть открывающихся возможностей перед разработчиком. Важным является лишь то, что теперь разработчик не связан требованиями стационарного расположения сервера и клиентских приложений. Он сам может распределять информационное пространство по БД и, более того, пользователь также может иметь возможность перераспределять структуру при изменении предметной области.

Система взаимосвязи сервера БД и клиентских приложений также задаётся разработчиком информационной системы и может изменяться как по форме связи, начиная от передачи файлов и заканчивая прямым общением через IP-адрес и выделенный порт по

любому доступному протоколу, также и по структуре передаваемой информации, от обычной системы «запрос-ответ», до передачи целых БД-сателитов.

Литература

1. Микляев И.А., Концепция разработки матричной универсальной базы данных с поддержкой древовидной структуры единицы информации и её универсального приложения // Вестник Воронежского государственного университета. Серия: Системный анализ и информационные технологии. 2010. № 2., Воронеж, С. 101-108
2. Микляев И.А., Формализованное описание основной структуры представления данных матричной универсальной объектно-реляционной базы данных // Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика. 2011. № 1., Астр., С. 61-68

УДК 004.042, 004.272.44

ФОРМАЛИЗАЦИЯ DATAFLOW-МОДЕЛИ ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА[1,2]

Салибекян Сергей Михайлович, ст. преподаватель, Московский институт электроники и математики Национального исследовательского университета «Высшая школа экономики», Москва, salibek@yandex.ru

Панфилов Пётр Борисович, к.т.н., доцент, Министерство экономического развития, Москва, panfilov@miem.edu.ru

В статье представлены основные элементы разрабатываемого формального аппарата для описания и анализа вычислительных систем и приложений, создаваемых в dataflow-парадигме организации вычислительного процесса. Парадигма управления вычислениями потоком данных (dataflow-парадигма) более приспособлена к реализации параллельных вычислений, чем классическая controlflow-парадигма, предусматривающая управление вычислениями потоком инструкций. Кроме того, в dataflow-парадигме более естественно и просто реализуется объектный принцип построения приложений, т.к. на основе dataflow-модели вычислений появляется возможность создавать сложно структурированные описания объектов и моделировать их поведение. Тем не менее, до сих пор вычислительные системы и приложения на основе dataflow-модели вычислений не могут занять своего достойного места среди параллельных вычислительных систем и приложений, одной из причин чему является не только сложность в аппаратной реализации модели, но и недостаточная формальная проработка самой dataflow-модели вычислений. В данной работе предлагается вариант решения задачи формализации вычислительного процесса, построенного в dataflow-парадигме, при этом фокусом наших исследований является вариант dataflow-модели организации вычислений на основе предложенного авторами объектно-атрибутного (ОА) подхода к организации вычислительного процесса [1, 2].

Основой формального аппарата для описания вычислительного процесса в ОА-системе послужили теория конечных автоматов [3] и процессная сеть Кана [4]. Процессная сеть Кана была выбрана ввиду того, что она неплохо подходит к моделированию параллельных вычислительных систем (ВС) с управлением потоком данных (dataflow), хотя при данной модели и существуют определенные проблемы при описании совместной обработки данных несколькими вычислительными узлами в случае параллельной вычислительной архитектуры с общей оперативной памятью. Поэтому модель Кана пришлось дорабатывать. Переработки также потребовала и модель конечных автоматов, т.к. классический автомат не приспособлен к обработке сложно структурированных типов данных и тем более к работе с объектами. Также потребовалось разработать и формат данных для описания сложно структурированных информационных конструкций.

---

[1] Исследование осуществлено в рамках программы фундаментальных исследований НИУ ВШЭ в 2013 году

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

Предлагаемая здесь формальная модель dataflow-системы ОА-архитектуры представляет собой совокупность элементов любой природы (счетные и несчетные множества, списки, вектора, функции, вектора функций и т.д.), сгруппированных во множества, называемые ОА-автоматами, которые, собственно, и реализуют обработку данных. Согласно ОА-подхода, как это представлено, например, в работе [1], в прикладной (программной или аппаратной) ОА-системе ОА-автомат является функциональным устройством (ФУ), которое осуществляет обработку информации. Все данные в формальной ОА-модели представляются в виде информационных пар (ИП). ИП – это двойка $c=\{a, l\}$, где $a \in A$ (A - множество атрибутов), нагрузка $l \in L =\{\Re \cup nil \cup S \cup \Omega\}$, где L – множество данных в нагрузке, $\Re$ – множество рациональных чисел, nil – обозначение пустой нагрузки, Ω - множество адресов (номеров) глобальной памяти; $S \in \Sigma^*$ – множество цепочек символов, принадлежащих алфавиту Σ. Все ИП можно разделить на два класса, а именно: информационные (служат для описания каких-либо объектов) и милликоманды (применяются для обмена информаций между ОА-автоматами).

Описанию нашей ОА-автоматно-сетевой модели dataflow-вычислительного процесса предпошлем ряд базовых определений и утверждений.

Дадим определение **цепочки ИП** (в ОА-ВС цепочка ИП является информационной капсулой [1]) в нашей ОА-автоматно-сетевой модели.

Определение 1. η является **цепочкой ИП**, если выполняются следующие условия:

1. Если ω - пустая цепочка ИП (не содержит ни одной ИП), то ω - цепочка ИП;
2. Если γ - цепочка ИП и c – ИП, то γc (или $\gamma \cup c$) – цепочка ИП (где $\cup$ - знак объединения цепочек ИП);
3. η - цепочка ИП тогда и только тогда, когда она является таковой в силу (1) и (2).

Примечание: В цепочку ИП могут входить одинаковые ИП.

Индексом ИП в цепочке является порядковый номер ИП в цепочке, например, η_i, где $i = 1,2,\ldots |\eta|$, а $|\eta|$ - длина (количество ИП) цепочки η.

Для реализации обработки информационных конструкций введем следующее понятие эквивалентности цепочек.

Утверждение 1. **Цепочки ИП** $\eta 1$ и $\eta 2$ **эквивалентны** (т.е. $\eta 1=\eta 2$) в том случае, если выполняются следующие условия:

1. совпадают длины цепочек ИП $\eta 1$ и $\eta 2$, т.е. $|\eta 1| = |\eta 2|$
2. $\eta 1=\eta 2=\omega$ или $\eta 1,i=\eta 2,i$, $i=1,2,\ldots,|\eta 1|$.

Определим **операции над цепочками ИП**, чтобы иметь возможность их обработки.

Определение 2. Операция **конкатенации (объединения)**: если α и β - цепочки ИП, то цепочка ИП $\alpha\beta$ ($\alpha \cup \beta$) называется конкатенацией (сцеплением, объединением) цепочек ИП α и β. $\alpha\omega=\omega\alpha=\alpha$, где ω - пустая цепочка ИП.

Определение 3. Операция **пересечения по правилу ИЛИ**: если α и β - цепочки ИП, то цепочка ИП γ является пересечением цепочек ИП α и β ($\alpha \cap \beta$) в том случае, если она содержит в себе ИП, удовлетворяющие следующему условию:

$\gamma = \{a_i \in \alpha: \exists\, b_j \in \beta$, такое что, $a_i=b_j\}$, где $i=1,2,\ldots, |\alpha|$, $j=1,2,\ldots, |\beta|$.

Определение 4. Операция **пересечения по правилу И**: если α и β - цепочки ИП, то цепочка ИП γ является пересечением цепочек ИП α и β (α and β) в том случае, если она содержит в себе ИП, удовлетворяющие следующему условию:

если $\alpha \cap \beta = \alpha$, то $\gamma = \alpha$, иначе $\gamma = \omega$.

Т.е. если в цепочке β для каждой ИП из α найдется равная ей ИП, то γ будет содержать цепочку α, иначе γ будет содержать пустую цепочку ω.

Определение 5. **Связка цепочек** ИП (W) – это взаимосвязанная структура данных, состоящая из цепочек ИП. В ячейках связки W могут содержаться цепочки ИП $W=\{w_1,w_2,\ldots w_{|W|}\}$, где $|W|$ - длина цепочки цепочек символов, $w_1,\ldots,w_{|W|}$. Для индексации конкретной ИП в цепочке цепочек ИП можно пользоваться двойным индексом: W_{ij} – ИП, которая содержится в j-ой ИП i-ой цепочке ИП из W.

Связка цепочек ИП W будет использоваться в качестве общей памяти, в которой располагаются данные, к которым имеют доступ все вычислительные устройства.

Определение 6. Объектно-аттрибутный граф или **ОА-граф** представляет собой набор цепочек ИП (капсул), объединенных ссылками, расположенными в нагрузках ИП (рис. 1).

Цепочки ИП, расположенные в связке W, соединенные с помощью ссылок, помещенных в нагрузках ИП (где ссылка принадлежит множеству адресов глобальной памяти Ω), объединяются в **ОА-граф**, который можно использовать для описания сложно структурированных объектов, включая как данные, так и программы.

Для отображения элементов множества Ω на ИП, расположенные в общей памяти W, вводится функция **FromMem**, а для преобразования индексов ИП в W на множество Ω - функция **ToMem**.

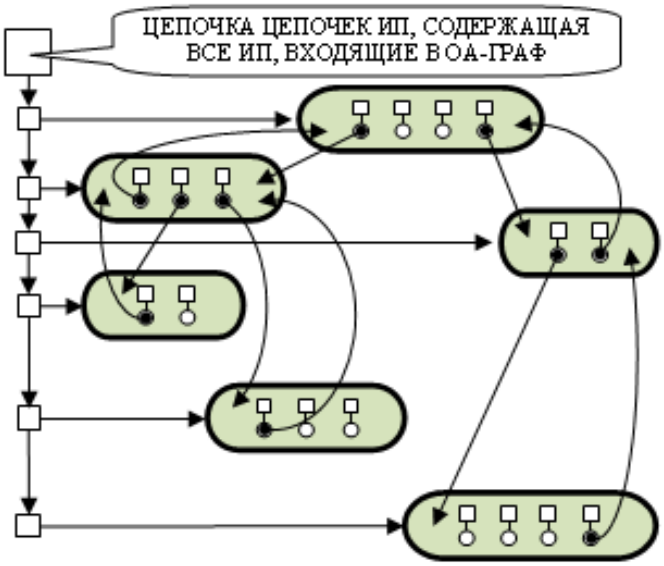


Рис. 1 - Объектно-атрибутный граф

Для удобства анализа ОА-графа составим связку цепочек ИП V, в которую будут входить все цепочки ИП, входящие в ОА-граф, и отныне будем считать, что ОА-граф – это связка таких цепочек ИП, что ссылки, расположенные в нагрузках ИП этих цепочек, указывают только на ИП, входящие в состав ОА-графа, т.е.:

$\forall\ c=\{a,l\} \in V,\ l \in \Omega : FromMem(l) \in V$, где V – множество всех ИП в ОА-графе.

Благодаря этому введению появляется возможность сравнения ОА-графов.

Дадим еще одно определение, необходимое нам для осуществления сравнения ОА-графов.

Определение 7. **Переиндексацией ОА-графа** будем называть изменение последовательности цепочек ИП в V.

Также введем функцию индексации памяти **IdMem**, которая задает однозначное отображение элементов множества адресов $\omega_i \in \Omega$ и $\omega_i \in Load(V_{ij})$, которые встречаются в ОА-графе, на множество индексов I (отображение является инъекцией). И теперь во время сравнения ИП, в том случае, когда у сравниваемых ИП нагрузки есть l_1, $l_2 \in \Omega$, эти нагрузки будут равны в том случае, когда $IdMem(l_1)= IdMem(l_2)$.

Утверждение 2. **ОА-графы G1 и G2 эквивалентны**, если можно подобрать такую переиндексацию G1 и такие функции индексации памяти IdMem1 и IdMem2 для G1 и G2, соответственно, что выполняются следующие условия:

1. $|G1|=|G2|$;
2. $\forall\ i \in 1,2,\ldots|G1|,\ G1_i=G2_i$.

Утверждение 3. **АО-графы G1 и G2 изоморфны (G1 ≈ G2)**, если выполняются следующие условия:

1. $|G1|=|G2|$;

2. Можно сделать такую переиндексацию ОА-графа и подобрать такие функции индексации памяти IdMem1 для G1 и IdMem2 для G2, что $\forall$ i $\in$ 1,2,…| G1|, $G1_i$=$G2_i$.

Если G1 и G2 изоморфны, то это значит, что оба графа описывают один и тот же объект.

Утверждение 4. **ОА-граф G1 частично изоморфен ОА-графу G2 (G1 ≈> G2)**, если можно подобрать такую переиндексацию G1 и такие функции индексации памяти для графов G1и G2, что $G1_i \cap G2_i = G1_i$, где i=1,…|G1|.

Частичная изоморфность означает, что объект G1 входит в состав объекта G2.

Теперь представим обобщенное описание работы ОА-автоматной сети как модели dataflow-вычислительного процесса.

Обработчиком информации, представленной в виде ИП, является ОА-автомат. В отличие от классического автомата, его состояние задается не множеством состояний, а контекстом K. **Контекст ОА-автомата K** – это набор сущностей любой природы (скалярные или векторные величины, счетные или непрерывные множества и т.д.). Если представить каждую сущность, входящую в контекст ОА-автомата, как измерение в многомерном пространстве, то состояние автомата будет представлять собой точку в этом пространстве. Изменение состояния автомата задается функцией изменения состояния автомата F:

$$K'=F(K),$$

где F – функция преобразования состояния K в новое состояние ОА-автомата K′.

Функцию F можно задать с помощью вектора предикатов переходов $\overline{P}$ и вектора функций перехода ($\overline{F}$). При этом размерности векторов должны совпадать, т.е. $|\overline{P}| = |\overline{F}|$. Аргументами предиката P_i и функции преобразования элемента контекста F_i, где i=1,2,…, $|\overline{P}|$, являются все параметры, входящие в контекст K. F_i представляет собой вектор функций, где каждая функция возвращает значение одного из параметров контекста $\overline{F}_i=[F_{i1},F_{i2},\ldots,F_{i|K|}]$, $K_j'=F_j(K)$, где j=1...|K|.

Отображение состояния контекста K на новое состояние K′ осуществляется следующим образом: происходит перебор предикатов из вектора предикатов и, если $P_i(K)$=true, то $K'=F_i(K)$ (т.е. $K_j=F_{ij}(K)$, где i=1,2,..,$|\overline{F}|$, j=1,2,…, |K|) и перебор заканчивается. В том случае, когда ни один из предикатов P_i не принимает истинного значения, контекст ОА-автомата остается неизменным. Следует отметить, что вектора $\overline{F}$ и $\overline{P}$ входят в контекст ОА-автомата K и поэтому могут быть изменены вместе со всем контекстом. Функция F входит в контекст ОА-автомата и также может быть изменена после перехода ОА-автомата в новое состояние. Таким образом, алгоритм работы (поведение) автомата может изменяться непосредственно во время вычислительного процесса.

ОА-автоматы делятся на несколько типов, каждый из которых реализует определенный функционал и имеет свой индивидуальный контекст. Однако контекст ОА-автомата любого типа обязательно включает в себя входную и выходную очереди милликоманд. Так, функция изменения контекста F может быть активизирована только в том случае, когда во входной очереди милликоманд присутствует хотя бы одна милликоманда (после активации функции F милликоманда из головы очереди милликоманд удаляется). Во время изменения своего контекста ОА-автомат может помещать информацию в конец своей выходной очереди милликоманд и тем самым передавать данные другим ОА-автоматам, для которых эта очередь является входной. При этом одна очередь милликоманд может входить в контекст одного ОА-автомата в качестве выходной и в контекст другого ОА-автомата в качестве входной очереди. В контекст любого типа ОА-автомата также в обязательном порядке входит общая оперативная память W. ОА-автоматы, таким образом, получают возможность работы с данными, расположенными в ней. Передача милликоманд между ОА-автоматами может происходить через автомат-коммутатор типа «Шина» (Bus) или напрямую.

Определение 8. Совокупность взаимодействующих ОА-автоматов будем называть **ОА-сетью**.

Структура ОА-автоматной сети, описывающей параллельный dataflow-вычислительный процесс, приведена на рис. 2.

Определение 9. **Выполнением S ОА-сети** будем называть последовательность активаций функций изменения контекстов ОА-автоматов: $S=F_{i1},F_{i2}\ldots$.

Примечание: В один момент может быть активирована только одна функция.

Функция перехода Fi, где i есть номер ОА-автомата, считается разрешенной к активации в том случае, если во входной очереди ОА-автомата присутствует хотя бы одна милликоманда. Выполнение ОА-сети будем считать завершенным в том случае, когда входные очереди всех ОА-автоматов пусты.

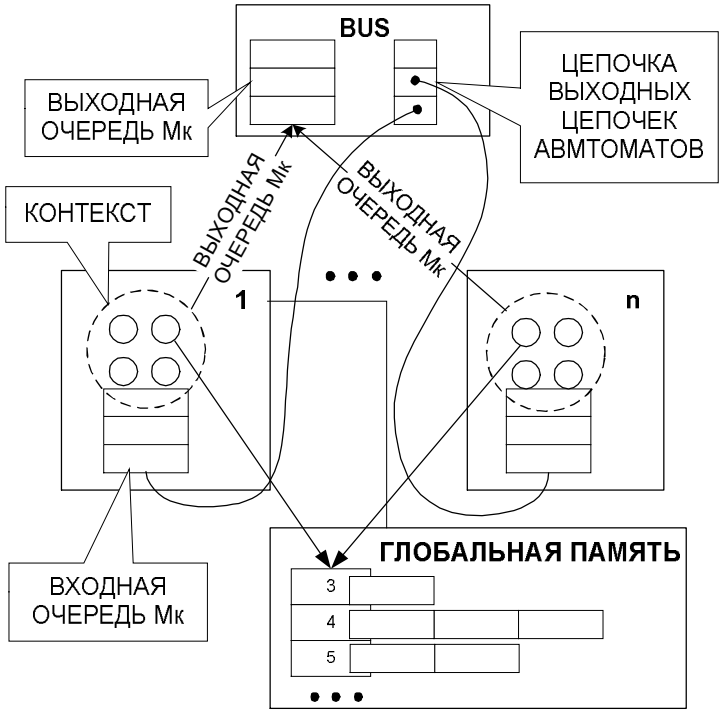


Рис. 2 - Сеть ОА-автоматов

Применение предложенного ОА-сетевого формализма описания dataflow-вычислений можно продемонстрировать на примере параллельного приложения семантического разбора естественного языка (англ. Natural Language Processing, NLP).

В последнее время данное научное направление привлекает большое внимание научной общественности. Во-первых, семантический анализ востребован во многих областях: web-поиск, машинный перевод, филология и т.д. Во-вторых, в данной области остается много нерешенных проблем, включая полноценный семантический анализ естественного языка, для реализации которого необходимо разработать робастую методику семантического разбора языка. Существующие подходы, например, на основе теории конечных автоматов, теории фреймов или концепции «смысл-текст» [5-7], не могут похвастаться надежностью и универсальностью. ОА-сетевой подход лучше всего подходит для решения такой задачи и имеет ряд преимуществ, включая возможность реализации объектного принципа, эффективное распараллеливание вычислений, реализацию на распределенных ВС и др.

В состав ОА-сети для реализации вычислительного процесса семантического анализа языка должны входить ОА-автоматы (функциональные устройства) следующих типов:

- **Lexica** осуществляет обработку входной последовательности символов, выделение из нее лексем (лексический анализ), формирование ИП с описанием лексем для дальнейшей обработки.
- **IcManager** отвечает формирование и модификацию новых ИП, цепочек ИП и ОА-графов.

- **List** – это автомат для сравнения капсулы с капсулами, находящимися в списке капсул, и выполнения последовательности действий в зависимости от результата сравнения.
- **Find** – это автомат для сравнения капсулы с эталонной капсулой и выполнения последовательности действий в зависимости от результата сравнения.

Опишем более подробно автомат Find, которому передаются три цепочки, включая эталонную цепочку ИП, последовательность милликоманд для выполнения в случае совпадения сравниваемой цепочки с эталонной и последовательность милликоманд для реализации при несовпадении цепочки с эталоном.

Контекст автомата Find будет представлять собой следующую структуру:

$$AF=\{A, L, MkIn, MkOut, F\cup T,\eta s,\eta f\},$$

где A и L – это множество атрибутов милликоманд и множество данных в нагрузке ИП;

MkIn и MkOut – очереди входных и выходных милликоманд, соответственно;

F – функция изменения контекста;

T – эталонная капсула;

ηs и ηf – капсулы, передаваемые в выходную очередь милликоманд, в случаях, когда пересечение входной капсулы с эталонной $\neq \omega$ (success) и $= \omega$ (failure), соответственно.

Для управления автоматом Find используются следующие **атрибуты милликоманд:**

Set – установить эталонную цепочку ИП;

SuccessProgSet и FailProgSet – установить цепочку ИП с программой, передаваемой в выходную очередь милликоманд в случае, если в результате сравнения с эталонной цепочкой ИП получилась непустая (success) и пустая цепочка ИП ω (failure), соответственно;

FindOr и FindAnd – сравнение по правилу ИЛИ и по правилу И, соответственно, эталонной цепочки ИП с цепочкой ИП из нагрузки.

Правила изменения контекста будут выглядеть следующим образом (справа от знака «$\rightarrow$» находится предикат перехода, слева – функция изменения контекста ОА-автомата):

F:

F_1: MkIn=Set $\rightarrow$ T=Load(MkIn)

F_2: MkIn= FailProgSet $\rightarrow$ ηf =Load(MkIn)

F_3: MkIn= SuccessProgSet $\rightarrow$ ηs =Load(MkIn)

F_4: MkIn= FindOr, Load(MkIn) $\cap$ T $\neq \omega \rightarrow$ MkOut= MkOut $\cup$ ηf

F_5: MkIn= FindOr, Load(MkIn) $\cap$ T $= \omega \rightarrow$ MkOut= MkOut $\cup$ ηs

F_6: MkIn= FindAnd, Load(MkIn) and T $\neq \omega \rightarrow$ MkOut= MkOut $\cup$ ηf

F_7: MkIn= FindAnd, Load(MkIn) and T $= \omega \rightarrow$ MkOut= MkOut $\cup$ ηs

Семантический анализ языка в ОА-сети происходит следующим образом. Распознаваемый текст поступает на ОА-автомат Lexica, который выделяет из исходного текста лексемы и оформляет их в виде ИП. Далее лексемы обрабатываются автоматами Find и List, а автомат IcManager формирует ОА-графы. Затем на основе запроса пользователя на анализируемом языке формируется ОА-граф запроса и производится определение частичной изоморфности ОА-графа запроса ОА-графу, описывающему смысл текста.

Предлагаемый формальный аппарат описания dataflow-вычислительного процесса ОА-архитектуры позволит в дальнейшем проводить математическое моделирование и анализ распределенных вычислительных систем и приложений, реализуемых в dataflow-парадигме. Модель выглядит более универсальной, чем такие часто применяемые модели параллельных ВС, как автоматные, на основе сетей Петри или процессных сетей Кана, т.к. позволяет моделировать параллельный вычислительный процесс, dataflow- и объектные системы.

Работы по формализации ОА-подхода к реализации распределенных вычислительных систем и приложений ведутся в рамках тематической НИР при финансовой поддержке Центра фундаментальных исследований НИУ ВШЭ.

Литература

1. Салибекян С.М., Панфилов П.Б. Объектно-атрибутная архитектура – новый подход к созданию объектных систем // *Информационные технологии*.—2012.—No. 2 (186).—С.8-13.
2. Салибекян С.М., Панфилов П.Б. Моделирование суперкомпьютерной вычислительной системы объектно-атрибутной архитектуры с управлением потоком данных // *Информационные технологии и вычислительные системы*.—2013.—No. 1.—С.3-10.
3. Брауэр В. М. Введение в теорию конечных автоматов: Радио и связь, 1987. — 392 с
4. Ключев А.О., Кустарев П.В., Ковязина Д.Р., Петров Е.В. Программное обеспечение встроенных вычислительные систем. – СПб.: СПбГУ ИТМО, 2009. – 212 с. http://window.edu.ru/resource/411/63411/files/itmo368.pdf
5. М. Минский. Фреймы для представления знаний. / Пер. с англ. О.Н. Гринбаума; под ред. Ф.М. Кулакова. М. Энергия. 1979.
6. Мельчук И.А. Опыт теории лингвистических моделей «СМЫСЛ <–>ТЕКСТ» - М.: Школа «Языки русской литературы», 1999.
7. Нгуен Ба Нгок, А.Ф. Тузовский. Обзор подходов семантического поиска // Доклады томского государственного университета систем управления и радиоэлектроники. Томский государственный университет систем управления и радиоэлектроники. Томск. 2010. URL: http://www.tusur.ru/filearchive/reports-magazine/2010-2-2/234.pdf

УДК 004.4

ПРИНЦИПЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОЕКТИРОВАНИЯ В ФОРМИРОВАНИИ АДАПТИРУЕМЫХ ИНФОРМАЦИОННЫХ СРЕД[1,2]

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

Введение

Современные информационные системы вырастают и приобретают глобальный характер и охватывают широкий спектр функций. В связи с чем актуальным стал вопрос разработки информационных систем, обладающих свойством адаптируемости, т.е. способности к совершенствованию, модификации, устойчивыми к изменениям среды функционирования или функциональных требований. При этом система должна предусматривать добавление новых компонентов по мере их разработки минимальными усилиями. Применение принципов объектно-ориентированного проектирования в разработке таких систем позволяет достигнуть перечисленных свойств.

Рассмотрим механизмы объектно-ориентированного проектирования в разработке информационно-вычислительной среды управления ресурсами исследовательского пространства. Под исследовательским пространством (ИП) понимается информационно-вычислительная среда, обеспечивающая комплексную поддержку исследовательской деятельности. Основная цель построения исследовательского пространства – предоставление через Интернет сервисов в области научных исследований на базе информационно-коммуникационных технологий для представителей научных сообществ, образования и бизнеса. Вторая цель - консолидация и концентрация ресурсов исследовательской деятельности в единое научное информационно-вычислительное пространство, которое включает ресурсы и достижения в области науки и формируется совместно владельцами

[1] Работа выполнялась в рамках ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009-2013 по теме «Разработка модели автоматизированной системы интеграции открытых виртуальных лабораторных комплексов» государственного контракта номер 02.740.11. 0658 от 29.03.2010.
[2] Лауреат номинации "**Лучший доклад по UML-моделированию**". Авторы доклада награждаются книгой Иванова Д.Ю. и Новикова Ф.А. "**Моделирование на UML. Теория, практика, видеокурс**" (www.umlmanual.ru) с автографами авторов

(авторами) из различных научных организаций и исследовательских учреждений. К ресурсам исследовательской деятельности относятся информационные ресурсы в составе: интеллектуальных (математические модели объектов и методы их исследования, результаты численных экспериментов над объектом, отчеты и публикации), алгоритмические (программы и программные комплексы, осуществляющие вычислительный эксперимент), такие ресурсы назовем виртуальными лабораториями (ВЛ); организационные (правила и инструкции выполнения исследования, нормативная документация). Также в состав ИП входят вычислительные ресурсы (ВР), аппаратные комплексы, необходимые для выполнения экспериментов [1].

В основу формирования исследовательского пространства положен принцип распределенного наполнения и централизованного управления ресурсами. Таким образом, управление ресурсами предполагает выполнение следующих процессов, описанных в виде диаграммы прецендентов (рисунок 1):

1. Формирование заявки на включение ВЛ в ИП. Процесс заключается в заполнении заявки установленной формы на сайте системы о желании подключить ресурс, например виртуальную лабораторию в состав ИП. В заявке указываются условия предоставления ВЛ в составе ИП и ее описание.
2. Рассмотрение заявок на регистрацию ВЛ. Процесс выполняется актером «Администратор ИП», который, рассмотрев заявку, может либо отклонить ее, либо одобрить. В случае одобрения заявки выполняется регистрация сведений о заявленной ВЛ и об ее разработчике. Также на основании шаблонов формируется список требований к зарегистрированной ВЛ.
3. Метрологические испытания ВЛ (закрытое тестирование)/ Открытое тестирование ВЛ. Процесс выполняется актером «Эксперт», «Открытое тестирование ВЛ» – любым пользователем ВЛ и заключается в определении соответствия предлагаемой ВЛ заявленным требованиям в соответствии с алгоритмом тестирования.
4. Формирование заключения о возможности включения ВЛ в состав ИП. По результатам тестирования экспертом выдается заключение о возможности подключения ВЛ в состав ИП. В случае положительного заключения формируется список требований для настройки интерфейса ВЛ с ИП, формирование требований к профилю ресурса для взаимодействия с пользователем.
5. Регистрация ВЛ. В случае положительного заключения экспертов администратор размещает ВЛ в составе ИП.
6. Настройка интерфейса АС и ВЛ с целью предоставления доступа ресурсу удаленным пользователям через портал системы.

Механизмы обеспечения адаптируемости информационно-вычислительной среды

В основу проектирования исследовательского пространства положены принципы объектно-ориентированного проектирования, которые позволяют вносить изменения в информационную среду с минимальными усилиями, осуществлять динамическое наращивание функционала в соответствии с изменчивостью требований прикладной области научных исследований.

В соответствии с принципом объектной декомпозиции [2-3], ИП рассматривалось как набор взаимодействующих объектов, обменивающихся сообщениями. Для этого определены состав объектов, их структура и поведение. Это позволило спроектировать соответствующие классы и определить зависимости между ними, описанные в виде диаграммы классов (рис. 2).

Для локализации изменений проведен анализ классов на предмет устойчивости к изменениям. Принцип «локализации» изменений позволяет выявить классы, обладающие стабильностью и классы, которые будут подвержены изменениям. Такое разделение подразумевает существование компонент, находящихся под непосредственным воздействием изменений (речь идет о тех классах, чьи обязанности при внесении изменений

корректируются), и компонент, подпадающих под это воздействие косвенно (это те классы, чьи обязанности остаются без изменений, но реализация меняется в соответствии с коррективами, внесенными в классы первой группы).

Проектирование операций классов осуществлялось с расчетом на ограничение области распространения ожидаемых изменений. Таким образом, определены неустойчивые классы с учетом прогнозируемых изменений объектов прикладной области, определены зависимости между ними:

- виртуальные лаборатории (класс ВЛ)
- вычислительные ресурсы (класс ВР)
- алгоритмы испытаний (класс Алгоритмы испытаний)
- требования (Класс Требования к заявке).

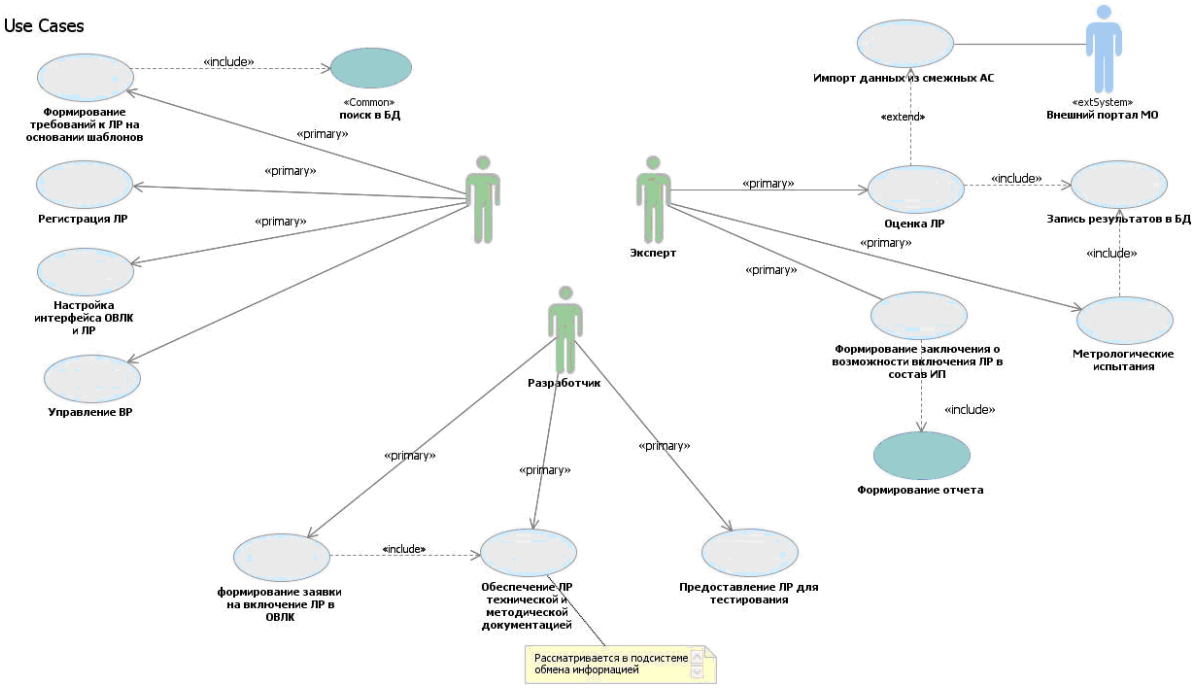


Рис. 1 - диаграммы прецендентов USE-CASE Управления ИП

Экземпляры этих классов подвержены изменениям в связи с развитием методик проведения экспертизы ресурсов, а также с развитием самого ресурса и возможного изменения его поведения. Вместе с тем, к стабильным классам относятся классификаторы и справочники:

- объекты исследования,
- методы исследования,
- организации,
- пользователи,
- шаблоны требований и заключений.

Для снижения зависимости между неустойчивыми и устойчивыми классами проводился анализ отношений и определения типа ассоциаций. При этом соблюдался принцип минимизации силы связности между ними. Известно, что наиболее строгая зависимость – это композиция, наименее слабая – обобщение [2]. С учетом выше сказанного, стояла задача минимизировать зависимости типа композиция. Таким образом, установлены отношения для классов ЛР и ВЛ и ВР –обобщение. Эта зависимость позволяет обобщить атрибуты и операции в классе ЛР с одной стороны, и наследовать их в классах ВР и ВЛ, что позволить повторно использовать методы в наследуемых классах. С другой стороны, изменчивость классов ВР и ВР позволяет учитывать индивидуальные атрибуты и операции для этих классов и вносить изменения с учетом изменчивости требований. Для классов ВЛ, Объект исследования и Метод исследования установлено отношение агрегация,

т.к. с одной стороны класс ВЛ использует данные этих классов, вместе с тем между ними нет строгой зависимости. Для всех остальных классов установлено отношение ассоциации, обладающей достаточно слабой связностью, что позволяет уменьшить зависимость между классами. Проведение всестороннего системного анализа классов и зависимостей между ними необходимо на этапе проектирования и учитывается при реализации системы.

Следующим принципом обеспечения модифицируемости является повторность использования. Принцип подразумевает выявление типовых функций и их реализация определенным набором классов, которые являются наиболее стабильными по отношению к изменчивости.

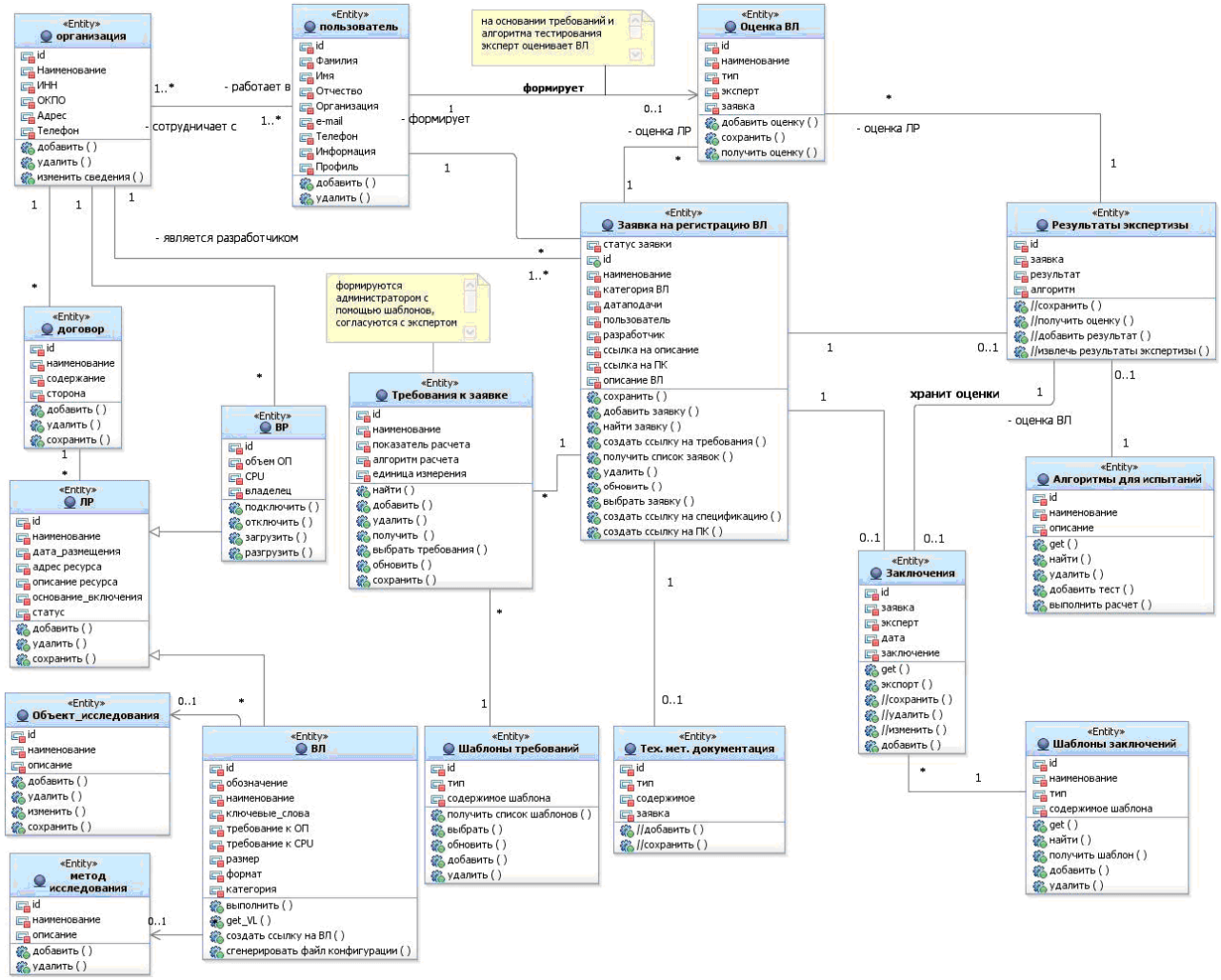


Рис. 2 – Диаграмма классов Управления ИП

Вместе с тем, повторность использования допустима и для прикладных сервисов, предоставляемых ИП:

- сервисы управления файлами (выбор файла, генерация имени файла, загрузка файла, проверка и обработка исключительных ситуаций)
- сервисы управления документами (выбор шаблона документа, сохранение документа)
- сервисы обработки строго типизированных XML–файлов (чтение параметров, запись параметров).

Обеспечение независимости разработки сложных приложений, к которым относится ИП, достигается путем декомпозиции системы на подсистемы, с минимизацией зависимостей друг от друга. Структурно система включает следующие блоки:

- блок регистрации ресурсов, отвечающий за их создание, поиск и фильтрацию, подключение в состав информационного пространства, блок учитывает интеллектуальные и вычислительные ресурсы;

- блок управления заявками, отвечающий за регистрацию и учет заявок на размещение ресурсов в составе информационного пространства;
- блок верификации и сертификации интеллектуальных ресурсов, отвечающий за оценку соответствия программы ЭВМ техническим и технологическим регламентам.

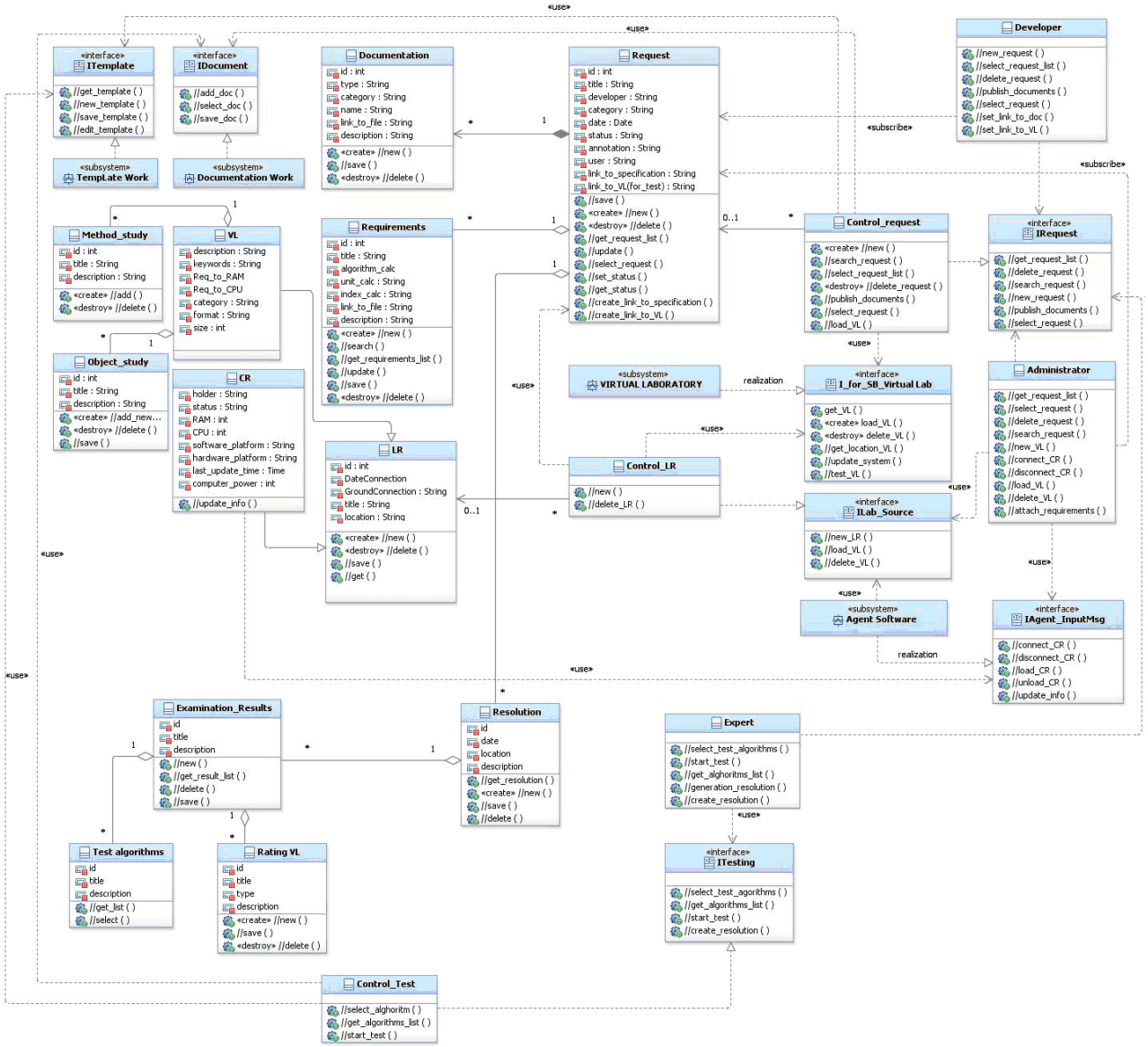


Рис. 3 - Диаграмма классов с описанием интерфейсов

Таблица 1 - Описание интерфейсов системы

Интерфейс	Назначение	Подсистема, предоставляющая интерфейс	Реализующий класс
ILabSource	описание операции, реализуемых классами лабораторных ресурсов	Lab Sources Maintain (Регистрация ЛР)	Control_CR, Control_VL
IRequest	описание операции, реализуемых классом заявок на регистрацию ресурса	Order of Lab Sources Managment (Управление заявками)	Control_request
ITesting	описание операции, реализуемых классом тестирования ресурсов	Lab Sources Testing (Тестирования ВЛ)	Control_ Test

Для взаимодействия с классами подсистемам получены следующие интерфейсы (таблица 1).

На рисунке 3 представлена диаграмма классов с описанием интерфейсов и уточненных ассоциаций между классами и с классами внешних подсистем посредством интерфейсов.

Выводы

Анализ функций управления ресурсами ИП позволил выделить объекты и спроектировать классы с учетом их устойчивости по отношению к изменениям. В соответствии с методологией объектно-ориентированного проектирования идентифицированы связи между классами в рамках функций управления ресурсами ИП. Выявление существующих связей между классами позволило их спроектировать с учетом свойств стабильности и обеспечить адаптируемость системы. Применение интерфейсов для взаимодействия компонентов подсистем обеспечивает возможность устранения волновых эффектов при внесении изменений и возможность расширения функционала системы.

Представленные механизмы объектно-ориентированного проектирования позволяют вносить расширения в систему, наращивать функционал с учетом индивидуальных потребностей научных сообществ и изменчивости требований. Адаптируемость таких систем значительно сокращает сроки и стоимость внедрения в решении задач научных исследований, а также сокращает затраты на их эксплуатацию.

Литература

1. И.Н.Ефимов, С.Ж.Козлова, С.А. Жукова Концептуальные основы интеграции открытых виртуальных лабораторных комплексов. - Вестник ИжГТУ. - № 3, 2011
2. Иванов Д. Ю. и Новиков Ф.А. "Моделирование на UML. Теория, практика, видеокурс". - Профессиональная литература, Наука и техника. – 2010. – с. 640
3. Арлоу, Нейштадт. UML 2 и Унифицированный процесс: практический объектно-ориентированный анализ и проектирование. - Символ-Плюс, 2-е изд. - 2007. – с. 624

УДК 6581.512

ИНТЕЛЛЕКТУАЛЬНАЯ ОБЪЕКТНО-ОРИЕНТИРОВАННАЯ ИМИТАЦИОННАЯ МОДЕЛЬ ПРОИЗВОДСТВЕННОЙ СИСТЕМЫ

Новиков Николай Иванович, к.т.н., доцент, Уфимский государственный авиационный технический университет, филиал в г. Кумертау, Россия, Кумертау, oka_novikov@mail.ru

Автоматизированные механообрабатывающие участки являются сложными системами, включающими разнородное технологическое оборудование, транспортно-складские системы и др. Влияние различных внутренних и внешних возмущений, действующих на элементы механообрабатывающего участка затрудняет достижение проектных показателей.

Обеспечение устойчивости функционирования производственных систем в таких условиях является актуальной и сложной задачей.

В задачах повышения устойчивости функционирования производственных участков возникают проблемы как качественного, так и количественного характера. При традиционном изучении сложных систем, какими являются производственные участки, удельный вес качественных показателей преобладает. Использование имитационного моделирования для исследования функционирования производственных участков позволяет анализировать их с использованием количественных характеристик изменения процесса функционирования участков, вызванных различными возмущающими факторами.

Однако необходимо отметить, что постоянно возрастающая сложность производственных участков, слабая степень формализации организационно-технологических решений не позволяет исключить качественные оценки в задачах анализа и повышения устойчивости функционирования производственных участков.

Традиционные математические модели, используемые при организационно-технологическом проектировании для группового производства, являются во многих случаях слишком жёсткими и не позволяют в полной мере учесть разнообразие факторов, влияющих на логику функционирования решений. Опыт и традиции производства при этом остаются вне рамок используемых моделей и учитываются лишь на стадии функционирования

производственной системы.

Одним из перспективных направлений в этой отрасли является объектно-ориентированный подход, заявленный как перспективная методологическая основа проектирования сложных систем преимущественно в области программного обеспечения. Объектно-ориентированный подход - это методология, основанная на представлении модели системы, состоящей из абстрактных объектов и объединяющих их отношений. Считается, что такая методология отражает особенности человеческого мышления и его способности к классификации для уменьшения сложности восприятия окружающего мира [1].

Имитационная модель производственной системы разработана на основе объектно-ориентированного подхода и методов искуственного интеллекта или так называемого экспертного подхода. Такой подход более предпочтителен по отношению к существующим, которые основаны на жестких алгоритмах.

Для создания адекватной модели реального механообрабатывающего участка в рамках объектно-ориентированного подхода при построении программного обеспечения системы моделирования применены специальные механизмы. Такие механизмы, объединяющие свойства фреймов и семантических сетей, и базирующиеся на фундаментальных понятиях объектно-ориентированного подхода, созданы в процессе исследований и используются в программном обеспечении системы моделирования [3].

При создании модели производственной системы вначале была разработана исходная имитационная модель, с помощью которой решались следующие задачи:

- формирование базы знаний экспертно-диагностической системы на основе обобщения результатов экспериментальных исследований и выявленных взаимосвязей:
- количественная проверка сформированной базы знаний.

Выбор экспериментального пути решения указанных задач определяется следующими обстоятельствами:

- необходимостью комплексного изучения процессов функционирования всей совокупности технологических процессов в производственной среде и невозможностью их описания в аналитической форме;
- необходимостью рассмотрения динамических и стохастических характеристик процессов функционирования технологических процессов в производственной среде и влиянием различных возмущающих факторов, имеющих вероятностный характер;
- недостаточным уровнем первоначальных знаний о функционировании системы и взаимосвязях основных параметров (понимание законов взаимодействия параметров системы приходит в процессе решения задачи путем получения дополнительной информации об исследуемом объекте на основе использования модели).

В связи с этим целесообразно привести мнение Дж. Фон Неймана о том, что для сложной системы «на порядок труднее рассказать, что объект может делать, чем сделать сам объект»

[2, с.75], «... быстрее привести схему, чем дать описание всех ее функций и всех мыслимых обстоятельств» [2, с.67].

При разработке рабочей имитационной модели производственной системы в основу была взята базовая имитационная модель, применяющаяся при проведении экспериментальных исследований.

Программирование на языке Smalltalk является по самой своей природе таким, что позволяет последовательно наращивать возможности создаваемого продукта, т.е. Smalltalk хорошо приспособлен к разработке моделей пошагового усовершенствования. Такое свойство имеет два важных следствия:

- происходит преднамеренное смешение ролей программиста и пользователя, поскольку для них используется одна и та же метафора;
- становится возможным внутренне перемешивать в принципе различные виды деятельности программиста: программирование и отладку.

Это означает, что пользователь программы постепенно сможет перейти от простого использования к модификации и расширению программы посредством такого же взаимодействия с системой.

Выполнение программы, разработанной на объектно-ориентированном языке, - это последовательное выполнение посылок сообщений объектам. Во время выполнения определяется, какой объект получил то или иное сообщение. В классе этого объекта ищется метод, соответствующий данному имени сообщения. Если метод найден, то он выполняется, если нет - ищется в суперклассе, и далее по иерархии до самого старшего класса Объект, описывающего общие свойства всех объектов.

Определение и программирование классов (объектов) системы проведено в следующем порядке:

- выделен описываемый объект из множества объектов:
- определены свойства и связи этих объектов:
- определены диапазоны и характеристики изменения свойств и связей объекта (задание имен переменных класса и экземпляра);
- определены множества возможных методов поведения объекта (специфицирование поведения объектов данного класса);
- выполнено программирование методов, включая условия выполнения методов:
- выполнено тестирование на примере действующего участка.

Разработанное программное обеспечение имитационной модели состоит из 6 нижеуказанных основных классов, погруженных в объектно-ориентрованную среду Smalltalk:

- модель участка;
- продукция;
- ресурсы;
- стратегия;
- планировщик;
- окна просмотра системы планирования.

Взаимодействие объектов осуществляется под управлением объекта «Планировщик» путем посылки сообщений.

Для осуществления взаимодействия пользователя с системой был запрограммирован класс «Окна просмотра системы планирования» - графический интерфейс. Графический интерфейс предназначен для ввода и корректировки базы данных и знаний, отображения динамических состояний моделируемых объектов.

Перед началом моделирования пользователь подготавливает исходные данные по составу оборудования, режиму его работы на период моделирования с использованием окна «Оборудование» и «График работы». Оборудование в зависимости от специфики работы разделяется на: станки (механическая обработка), не механообрабатывающее (термическая, гальваническая обработка и др.). Для станков могут задаваться различные режимы работы.

В окне «Оборудование» имеется графическая интерпретация графика работы выделенного оборудования.

Информация о номенклатуре деталей, закрепленных за участком, задается при помощи окна «детали». С каждой позицией этого массива связан базовый технологический процесс и его варианты. Для конкретного планового периода из массивов: «детали», «технологические процессы» и «варианты технологических процессов» формируется конкретный рабочий массив.

Заключительном этапом подготовки исходной информации является задание параметров стратегий изготовления партий деталей.

Параметры стратегии изготовления партии деталей могут задаваться директивно или непосредственно пользователем. При этом задаются следующие параметры:

- начальная дата периода планирования;

- длительность периода моделирования;
- обозначение детали;
- номер партии конкретной детали;
- размер партии запуска детали;
- очередность запуска партий деталей в обработку;
- вид движения партий деталей;
- стратегия изготовления партии деталей;
- операция партии деталей, с которой начинается ее обработка;
- дата запуска партий деталей;
- операции технологического процесса конкретной партии деталей, которую планируется выполнить за период моделирования;
- срок окончания обработки партии деталей или операции.

Каждая партия деталей из заданной номенклатуры деталей может иметь свою стратегию изготовления. Для задания стратегии изготовления конкретной партии деталей предназначено «Окно стратегии».

По результатам анализа характеристик функционирования производственного участка стратегия может быть модифицирована и снова может быть проверена путем повторной имитации функционирования производственного участка.

При помощи указанного окна задаются следующие параметры для партии деталей:

- вид движения партий деталей (в системе предусмотрены следующие виды движения: последовательный, параллельный, смешанный);
- транспортная партия деталей (применяется при параллельном виде движения партии деталей);
- критерий выбора рационального варианта из множества возможных, в системе предусмотрено два критерия:
- минимальное пролеживание партии деталей и минимальное время завершения операции;
- замена базовой операции на альтернативную из возможного множества (имеется два режима: разрешается/запрещается замена операций);
- изменение технологического маршрута в случае невозможности реализации базового (имеется два режима: разрешается/запрещается замена операций);
- специализация рабочих мест–в системе предусмотрено два режима:
- разрешается/запрещается специализация рабочих мест;
- дробление партии запуска (имеется два режима: разрешается/запрещается дробление);
- предельное время ожидания (пролеживания) партии деталей перед рабочим местом, при превышении которого производится поиск альтернативных решений;
- коэффициент, отражающий относительные затраты на реализацию мероприятия (изменяются от 0 до 100).

Для визуализации результатов моделирования в системе запрограммировано несколько окон, например: график запуска-выпуска партий деталей, график движения партий деталей по рабочим местам, график движения выделенной партии деталей по рабочим местам, график загрузки выделенного станка и др., позволяющие пользователю анализировать результаты моделирования, выявлять узкие места и принимать решения по их устранению и проверять принятые решения.

При имитации функционирования производственного участка принятие решения о закреплении операции за рабочими местом осуществляется в соответствии с критерием (предельное время пролеживания партий деталей и минимальное время завершения). В случае, если фактическое время выполнения операции превышает заданное предельное значение, то производится поиск решений по локализации отказа.

Результаты моделирования отображаются в окне «график запуска-выпуска», при этом можно просматривать следующие параметры:

- обозначение детали прошедшей моделирование;
- количество деталей к запуску в обработку;
- номер партии и подпартий деталей;
- номер операции, с которой начинается обработка партии деталей;
- дата запуска партий и подпартий деталей в обработку;
- номер операции технологического процесса, которая выполнена последней в период моделирования;
- заданный срок окончания обработки партии детали;
- отклонение от заданного срока выпуска;
- суммарное время пролеживания партии детали.

В системе имеется также возможность отображения на экране монитора графика движения партий деталей по рабочим местам.

Для расширения возможностей системы при анализе ситуаций введено окно просмотра плана, позволяющее просматривать на экране монитора движение партий деталей по рабочим местам в табличной форме.

В окне также отмечаются места пролеживания партий деталей путем специальной метки.

В распоряжении пользователя имеется возможность упорядочивать график движения партий деталей по следующим признакам:

- по порядку поступления партий в обработку;
- по времени обработки партии по операциям;
- сортировка по группам оборудования.

«Окно обработки партии» предназначено для отображения на экране монитора графика движения конкретной партии деталей по рабочим местам в табличной форме.

В окне обработки партии деталей представляются следующие параметры:

- обозначение детали, для которой отображается движение партий деталей по рабочим местам;
- номер партии деталей, для которой отображается движение партии деталей по рабочим местам;
- номер операции технологического процесса;
- модель оборудования, за которым фактически была закреплена операция в процессе моделирования;
- время начала выполнения операции;
- время окончания выполнения операции;
- длительность выполнения операции на указанном оборудовании.

В окне предусмотрена возможность упорядочения массива:

- по времени начала операции технологического процесса;
- по порядку операций технологического процесса;
- по длительности обработки партий деталей.

Важнейшей особенностью созданной системы моделирования является применение эвристических знаний эксперта в качестве основной информации при описании объектов и имитации функционирования производственной системы. При разработке концепции системы моделирования учитывалось то, что роль эвристических знаний эксперта очень велика и практически невозможно предложить строгие алгоритмы принятия решений во всем многообразии возникающих производственных ситуаций. В связи с этим можно говорить лишь об использовании эвристик–правил поведения или действиях на основе анализа небольшого множества данных (фактов).

Организация процессов логического вывода на базе таких правил и является основой выработки решений экспертно-диагностической системы. В составе программного обеспечения системы моделирования используется процедурная компонента, осуществляющая логический вывод на основе базы правил, аккумулирующей эвристические

знания эксперта над базой данных вида сети фреймов, хранящей информацию о состоянии и поведении объектов производственной системы.

Листинг правила системы моделирования приведен на рисунке 1.

```
 «Перевод объема работ с подгруппы стФрезЧПУ на другой участок в случае
перегрузки»
< стЧПУ> # ГруппаОборудования
              ^фрейм имя Группы =#СтЧПУ;
              ЭфПринСтЧПУ: =^фреймэфФондПрин1;
< стФрезЧПУ> # ПодгруппаОборудования
             ^фрейм имя  =#СтФрезЧПУ;
             мПринФрезЧПУ: =^фрейм эфФондПрин.
< стФрезУнив> # ПодгруппаОборудования
             ^фрейм имя  =#СтФрезУнив;
             мПринФрезУнив: =^фрейм эфФондПрин.
< план> # План фреймПлана:= ^фрейм:
            ^фрейм имя  =# ПланГода;
            мПотрФрезУнив: =^фрейм текущаяТрудоемкостьПо: #фрезУнив:
            мПотрФрезЧПУ: =^фрейм текущаяТрудоемкостьПо: #фрезЧПУ:
«Если перегрузка менее 5% , то нет смысла применять правило»
{((мПотрФрезЧПУ - мПринФрезЧПУ)*100/ мПринФрезЧПУ) > 5);
«Если подгруппа стФрезУнив не имеет резерва свободной мощности, то
проверить -применялись ли правила R8, R1, R2, R22 »
1.05*(мПринФрезУнив - мПотрФрезУнив)  < (мПотрФрезЧПУ - мПотрФрезЧПУ)
оМощФрезЧПУ: = ((мПринФрезЧПУ - мПотрФрезЧПУ) abs):
кG: =(оМощФрезЧПУ// эфПринСтЧПУ) +1:
кG1: =(оМощФрезЧПУ// эфПринСтЧПУ) :
«кG - требуемое число станков»
мПринФрезУнив  мПринФрезЧПУ  мПотрФрезУнив  фреймПлана мПотрФрезЧПУ
оМошФрезЧПУ  кG  кG1  оМФрез  кМО  фреймМ  оMS  dMS  oFlaq   эфПринСтЧПУ
stalk {оМФрез:=0. oFlaq:=0  оМ1:= nil.
[оМФрез > 0 and: [оМФрез < 1000]]
whileFalse: [оМФрез: = Promter promt:  'имеется  ли возможность
использовать \ ', ' стФрезЧПУ другого участка и в каком объеме ?:
'printStrinq defaultExpression: '000.00'. (оМФрез < O oR:[оМФрез > 1000])
ifTrue : [Informate messaqe 'укажите объем в пределах 0...10000)
           withCrs backColor: 12].
if False : [(оМФрез=0)
               if True: [оМ1:=Promter promt: имеется ли возможность
использовать \' , ' стФрезЧПУ другого участка-купить. Взять, взять в
аренду?: printStrinq defaultExpression
ifFalse: [оMS: = 'снять объем и передать другому ', 'участку'.
                          оМ1=' нет'. dms: =оМФрез.].].]
"оМ1 =nil -нет вариантов: оМ1 = 'да' - аренда: оМ1='нет' -передача''
 (оМ1) =nil - «нет вариантов»
ifTrue : [фреймМ=кBase создать мероприятие.
             фреймМ название: ('правило не сработало' ):]
ifFalse: [(оМ1='да ') "аренда или покупка"
ifTrue:[фреймМ:=кBase создать мероприятие.
фреймМ название : ('необходима аренда или покупка'.]
ifFalse: ['возможен перевод их в др.участок ' .
фреймМ название :('перевод работ с ФрезЧПУ на другой участок'):
изменениеФрейма: фреймПлана
свойство : #коррТрудоемкости
ключ: #фрезЧПУ
изменение: оМ1 neqated:
сообщение: 'перенести объем работ на '  , оМ1 printStrinq,
                      'н/ч с стФрезЧПУ в ' , ' другой цех (участок)' ]]}
modify < план> ^фрейм: = фреймПлана.
make # Мероприятие ^фрейм:=фреймМ.
```

Рис.1 - Листинг правила имитационной модели производственной системы

В результате исследований разработана интеллектуальная система моделирования производственной системы, основанная на знаниях.

При разработке экспериментального образца имитационной модели производственной системы, основанной на знаниях, для сокращения длительности процесса создания и доработки программного обеспечения использован объектно-ориентированный подход к анализу и программированию. Это позволило значительно сократить сроки разработки и отладки модели, привлечь пользователя к созданию базы знаний имитационной модели на ранних стадиях создания модели.

Литература

1. Буч Г. Объектно-ориентированное проектирование с примерами применения: Пер. с англ.-М.: Конкорд, 1992.-519 с.
2. Нейман Дж.фон. Теория самовоспроизводящихся автоматов. – М.: Мир,1971 - 425с.
3. Новиков Н.И., Новиков В.Н. Применение имитационных моделей и экспертных систем при разработке производственных графиков // Казанская наука. Сборник научных статей. № 9 Вып.1. –Казань: Изд-во Казанский Издательский Дом, 2010.-С.186-190.

УДК 681.3

СИСТЕМА ПОДДЕРЖКИ РЕШЕНИЙ ПРИ ОРГАНИЗАЦИИ ГРУППОВОГО ПРОИЗВОДСТВА

Новиков Николай Иванович, к.т.н., доцент, Уфимский государственный авиационный технический университет, филиал в г. Кумертау, Россия, Кумертау, oka_novikov@mail.ru

Важнейшим направлением повышения эффективности машиностроительных предприятий с мелкосерийным и серийным типом производства является применение групповой обработки, что обеспечивает высокую производительность и гибкость производства.

Однако опыт применения групповой обработки в производстве показал, что часто потенциальные возможности этого подхода к организации подготовки и управления производством реализуются недостаточно полно. Анализ причин низкой эффективности групповых производств позволяет выделить следующие причины:

1. Жёсткая централизация процесса подготовки группового производства (группирование деталей, проектирование групповых технологических процессов изготовления деталей и управляющих программ для станков с ЧПУ, распределение детале/операций за рабочими местами), что приводит к тому, что решения, принятые на верхнем уровне, являются неточными и недостаточно полно учитывают особенности конкретных производственных условий, ориентированы на усреднённые условия производства.
2. Ограниченный («узкий») взгляд на систему проектирования и групповые технологические процессы, предусматривающий в основном решение вопросов точности и качества; вопросы реализуемости спроектированных групповых технологических процессов, вопросы проектирования технологических процессов в условиях неопределённости о производственной среде реализации технологических процессов практически не рассматриваются.
3. Отсутствие эффективных методов поддержки принятия решений на всех этапах жизненного цикла организационно–технологической подготовки группового производства. В существующих системах процесс принятия решений жёстко фиксирован. Системы проектирования не позволяют оперативно проводить изменения исходных организационно-технологических решений в связи с изменившимися условиями производства, а ориентированы на фиксированные состояния.
4. Системы поддержки принятия решений при организации группового производства не позволяют осуществлять оперативный поиск в конкретной производственной

ситуации компромиссов между групповой и индивидуальной технологией, между специализацией и универсализацией элементов производственного участка.

5. Имитационные модели используются для локальной проверки каких-либо проектных решений и не являются составной частью системы проектирования технологических процессов. Повышение устойчивости функционирования производственных участков с использованием моделей производится методом проб и ошибок, что требует значительных затрат времени на моделирование и снижает их эффективность.
7. Дестабилизирующее влияние большого количества производственных факторов и отсутствие методик по их выявлению и локализации.

Анализ решаемых задач технологической подготовки производства и управления групповым производством показал, что наиболее часто решаются задачи управления технологическими процессами и оборудованием в связи с необходимостью компенсации более частых по сравнению верхними иерархическими уровнями возмущений, соизмеримых по частотам с динамическими свойствами объекта управления. Для эффективного организационно-технологического проектирования на верхнем уровне управления необходимо знать все конкретные особенности производства: состояние парка оборудования, технологические процессы изготовления деталей, наличие рабочих и закрепление их за оборудованием, уметь анализировать информацию о ходе производства.

Таким образом, высокие требования к технологической подготовке группового производства со стороны постоянно развивающегося производства требуют совершенствования методов решения задач, связанных с реализацией групповых технологических процессов и оценкой адекватности проектных организационно-технологических решений.

В работе рассмотрены с системных позиций основные направления повышения эффективности организационно-технологического проектирования групповых автоматизированных производств.

Организационно-технологическое проектирование для групповых производств связано с использованием различных моделей (групп деталей, групповых технологических процессов, производственной системы, процессов организационно-технологического проектирования и др.). Эти модели должны представляться на разных стадиях жизненного цикла организационно-технологических решений в различной форме и с различной степенью детализации. При этом модели должны функционировать в единой программной среде.

Разнородность информации, неопределенность и неполнота исходных данных на каждом этапе технологической подготовки производства определяют сложность разработки информационно–математического и программного обеспечения. Для эффективной обработки информации в этих условиях средства автоматизации должны обладать широкими возможностями представления знаний смежных дисциплин, так как пользователями информации являются различные специалисты: технологи, нормировщики и мастера.

В этих условиях необходимо применение системы поддержки принятия решений, позволяющей представлять слабо формализуемые знаний экспертов, создавать базу знаний непосредственно специалистами предметной области.

В предлагаемой системе поддержки принятия решений мастера, диспетчера, технологи и другой персонал при подготовке и управлении производством обеспечиваются важной информацией, тактическими планами, и сведениями о приоритетах целей, а главное – принятие решений выполняется более быстро, точно и качественно, нежели с помощью существующих систем поддержки принятия решений.

Система поддержки принятия решений, базирующаяся на применении принципов искусственного интеллекта, помогает разрешать противоречия между технологическими и организационно-плановыми решениями, устранять конфликты взаимодействия, определять, какую информацию считать критически важной, и выдавать рекомендации по устранению узких мест, прекращать или изменять текущие действия, изменять приоритеты в обработке и другие параметры.

Структура системы поддержки принятия решений при организации группового производства приведена на рисунке 1.

В предлагаемой системе организационно-технологические решения до реализации в производстве проверяются с помощью имитационной модели производственной системы

Повышение эффективности организационно–технологического проектирования групповых производственных систем в условиях воздействия на систему внешних и внутренних возмущений возможно и целесообразно за счет избыточности структуры технологических процессов, а также элементов производственной системы. При этом избыточность структуры технологических процессов, а также элементов производственной системы должна быть предусмотрена заранее.

Необходимость использования избыточности объясняется следующим. Минимальное значение длительности производственного цикла изготовления партий деталей существует при определенном сочетании параметров и носит локальный характер. При этом незначительные изменения одного из параметров системы смещают минимум длительности производственного цикла в ту или иную сторону. В связи с этим локальные, несистемные решения, принимаемые для уменьшения длительности производственного цикла, позволяют стабилизировать функционирование участка только на короткий период времени, а иногда, наоборот, могут усугубить ситуацию.

Рис.1- Структура системы поддержки принятия решений при организации группового производства

Поэтому поддержание устойчивого функционирования участка в условиях действия внешних и внутренних возмущений существующими методами, не имеющими в своем составе средств для своевременного выявления возможных отклонений и принятия комплексных решений по их локализации, представляет определенные трудности.

Введение избыточности структуры технологического процесса, а также элементов производственного участка с целью повышения устойчивости функционирования производственного участка должно осуществляться при проектировании технологических процессов в максимальной степени. При этом использование при функционировании производственного участка должно быть гибким. Это обеспечивает в момент реализации минимальную, но достаточную избыточность структуры технологического процесса, а также элементов производственного участка. Частичное введение избыточности структуры

технологического процесса, а также элементов производственного участка и жесткое (без учета производственной ситуации) ее использование не даст значительного эффекта.

Введение избыточности структуры технологических процессов изготовления деталей возможно и целесообразно путем придания дополнительных структурных свойств (в данном случае адаптационных) технологическим процессам изготовления деталей.

Реализация принципа основана на введении в базу данных базового технологического процесса дополнительной (избыточной) информации о вариантах технологического маршрута и операций, вводимой на стадии проектирования технологических процессов изготовления деталей. При этом знания об условиях ветвления технологических процессов должны встраиваться в виде базы знаний в модель технологического процесса и использоваться при логическом анализе складывающихся производственных ситуаций и выборе из заданного множества адекватной структуры технологического процесса, исходя из производственной ситуации.

Для эффективного управления введенной избыточностью структуры технологического процесса, а также элементов производственного участка применена экспертно-диагностическая система.

Необходимость использования экспертно-диагностической системы обусловливается тем, что объективно существующая динамичность производственного участка не позволяет выполнить приемлемое аналитическое описание взаимосвязей между параметрами системы.

При этом для показателя устойчивости функционирования производственного участка важным является время с момента выхода показателя устойчивости за предельные значения до момента выхода из неустойчивого состояния. Это время должно быть минимальным. Эффективной можно считать систему, обеспечивающую минимальное время реакции системы на возмущение, с одной стороны, и, с другой стороны, позволяющую предсказывать приближение показателя устойчивости к предельному значению и выбирать регулирующие воздействия, препятствующие снижению устойчивости функционирования участка.

Такой подход объясняется тем, что традиционные методы технологической подготовки производства не способны работать в условиях слабой формализации задач. Поэтому наиболее адекватным подходом в этих условиях является подход на основе экспертных систем. В предлагаемой системе экспертные знания встраиваются в процессы принятия решений в виде базы знаний.

Это позволило эффективно решить поставленные в работе задачи анализа и формирования структур технологических процессов с позиций повышения устойчивости функционирования участка. Они требуют высокого уровня экспертиз, логического вывода и использования эвристики.

При формализации сложных процессов технологической подготовки группового производства возникают задачи, в которых число переменных и связей очень велико. В таких случаях задачи оптимизации не поддаются решению в лоб стандартными методами математического программирования.

Одним из направлений решения указанных проблем является применение объектно-ориентированной технологии анализа и разработки программного обеспечения, что позволяет снизить существующий барьер между аналитиками и разработчиками (дизайнерами и программистами), повысить надежность системы, упростить сопровождение, а также интеграцию с другими системами [1].

Основной идеей объектно-ориентированного подхода является использование языковых средств, которые на базе концепции абстрактных типов данных позволяют специфицировать новые классы программных объектов, образующих вычислительную среду, ориентированную на конкретную предметную область и позволяющие моделировать предметную область.

Ключевым при разработке системы поддержки принятия решений было проведение объектно-ориентированного системного анализа: формализация предметной области и

осмысление системы поддержки принятия решений как совокупности компонент. Системный анализ позволил:

- лучше понять «что надо делать»;
- упростить общение между участниками проекта (аналитики, разработчики, руководители, пользователи);
- отслеживать во времени изменения модели.

Была выполнена декомпозиция предметной области на базе фреймового подхода [2], что позволило подойти к анализу сложных систем с точки зрения иерархических структур данных или иерархий объектов со свойствами инкапсуляции, наследования и полиморфизма. Таким образом, объектно-ориентированный системный анализ позволил создать комплекс моделей, которые отображают не только данные и их взаимосвязи, но и методы обработки данных. Объектно-ориентированный анализ позволил получить более естественную формализацию системы в целом [3].

Применение объектно-ориентированной методологии анализа, проектирования и программирования позволило переходить от исходных требований к реализации создаваемой системы поддержки принятия решений при организации группового производства.

Использование объектно-ориентированного подхода позволило также решить проблемы сложности при создании системы поддержки принятия решений при организации группового производства.

При исследовании охватываемых проблемой объектов и процессов их взаимодействия использован системный подход и объектно-ориентированный подход, обеспечивающие необходимую глубину, комплексность за счет наличия функционального, морфологического и информационного аспектов исследования, методическое единство в решении отдельных задач проблемы, преемственность результатов на этапах анализа, проектирования, программирования и развития сложных программных систем.

На основе изучения особенностей автоматизированных производственных участков, методов организационно-технологической подготовки группового производства разработан комплекс объектно-ориентированных моделей, позволяющий исследовать и проектировать объект как единое целое.

Выполнена классификация объектов (классов), определение классов, определена форма определения, выделены наиболее важные связи между классами, сформирован словарь предметной области.

Для каждого фрейма - класса разработаны соответствующие методы.

Важной подсистемой в системе поддержки принятия решений является имитационная модель производственной системы. С помощью имитационной модели можно проверять организационно-технологические решения в различных режимах (детерминированный, стохастический).

При необходимости можно изменять различные входные параметры:

- коэффициент сменности оборудования;
- режим работы оборудования;
- срок поступления заготовок на участок;
- сроки запуска партий деталей в обработку;
- периодичность запуска партий деталей в обработку;
- очерёдность запуска партий деталей в обработку;
- структуру группового технологического процесса изготовления детали;
- дробление партий запуска деталей;
- изменение объёма выполняемых работ;
- изменение номенклатуры деталей производственного плана;
- вид движения партий деталей;
- специализацию рабочих мест.

В разработанной модели, в соответствии с принятым ранее объектно-ориентированным подходом к исследованию и программированию, знания о составе, структуре и поведении отдельных элементов производственного участка представлены в виде объектов, взаимодействующих между собой.

При этом методы функционирования каждого объекта в модели заложены в виде заранее определенных наборов методов. Набор действий по имитации производственных процессов описывается в терминах объектно-ориентированного подхода. На основе этой последовательности действий (сообщений) динамически формируется необходимая конфигурация оборудования, необходимая для обработки партий деталей

Для описания знаний о групповом производственном участке выбрана и программно реализована модель представления знаний, представленная в виде фрейма–класса в объектно-ориентированной среде.

При анализе предметной области выделены следующие фрейм - классы: оборудование участка, детали, изготавливаемые участком, партии запуска деталей в обработку, технологические процессы изготовления деталей и др.

В разработанной системе для повышения эффективности технологической подготовки группового производства вводится двухуровневая процедура формирования организационно-технологических решений.

На первом уровне формируются обобщенная группа деталей и групповой технологический процесс изготовления детали, который характеризуется высокой степенью избыточности структуры технологического процесса по отношению к базовому. Введение избыточности в технологические процессы объясняется невозможностью заранее сформировать ограниченное количество вариантов технологических процессов изготовления детали, удовлетворяющих в будущем конкретной производственной ситуации.

На втором уровне на этапе функционирования участка на базе обобщенного технологического процесса формируется оперативный технологический процесс изготовления партии деталей исходя из складывающейся производственной ситуации.

На каждом шаге процесса диагностирования производственной системы уменьшается неопределенность в связи с появлением дополнительной информации о функционировании производственного участка. Однако полностью устранить неопределенность практически невозможно. Поэтому в системе предусмотрен доступ пользователю, как лицу, принимающему решение (ЛПР). ЛПР на каждом шаге может вводить различные ограничения, приоритеты. Давать экспертные оценки, выбирать единственное решение из предлагаемого системой множества вариантов в случае, если эти варианты неразличимы по заранее заданным критериям.

В результате выполненных исследований разработана система поддержки принятия решений при организации группового производства.

Совершенствование системы поддержки принятия решений при организации группового производства выполнено с использованием объектно-ориентированного подхода и принципов искусственного интеллекта.

В предлагаемой системе поддержки принятия решений при организации группового производства используется комбинированный способ представления знаний: продукционно-фреймовый. Формализм представления инженерных знаний основан на использовании преимуществ фреймов и систем продукций. Фреймы положены в основу построения внешней языковой модели проблемной области и спецификации задач, а система продукций – в создание набора правил решения задач на базе фрейма.

Создание комплекса моделей системы поддержки принятия решений при организации группового производства на основе многоязыковой программной среды и объектно-ориентированного подхода, позволило наиболее адекватно описать реальные объекты и процессы.

Литература

1. Буч Г. Объектно-ориентированное проектирование с примерами применения: Пер. с англ.-М.: Конкорд, 1992.-519 с.
2. Минский М. Фреймы для представления знаний: Пер. с англ.-М.: Энергия, 1979.
3. Новиков Н.И. Объектно-ориентированный анализ процессов организационно-технологического проектирования для групповых автоматизированных производств // «Проблемы развития естественных, технических и социальных систем» Материалы международной научной конференции-часть 3 - Таганрог: Изд-во «Антон», ТТИ ЮФУ, 2007

Шахтинский институт (филиал)
Государственного бюджетного образовательного учреждения
высшего профессионального образования
"Южно-Российский государственный технический университет
(Новочеркасский политехнический институт)"

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

VIII Международная научно-практическая конференция

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

ОБЪЕКТНЫЕ СИСТЕМЫ – 2014

(с изданием сборника материалов конференции)

Информационные партнёры конференции:

Сообщество системных аналитиков

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

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

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

10–12 мая 2014 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., Заместитель директора по научно-исследовательской работе, Институт сферы обслуживания и предпринимательства (филиал) ДГТУ в г.Шахты, Россия, Шахты ***(председатель конференции)***
2. Олейник Павел Петрович, к.т.н., Системный архитектор программного обеспечения, Астон, Россия, Ростов-на-Дону ***(сопредседатель конференции)***
3. Божич Владимир Иванович, д.т.н., проф., Кафедра "Информационные системы и радиотехника", Институт сферы обслуживания и предпринимательства (филиал) ДГТУ в г.Шахты, Россия, Шахты
4. Сидельников Владимир Иванович, д.т.н., проф., Заведующий кафедрой "Экономика и прикладная математика", Педагогический Институт Южного Федерального университета, Россия, Ростов-на-Дону
5. Черкесова Эльвира Юрьевна, д.э.н., проф., Заведующая кафедрой "Управление персоналом, инновациями и качеством", Шахтинский институт (филиал), Южно-Российский государственный технический университет (Новочеркасский политехнический институт), Россия, Шахты
6. Михайлов Анатолий Александрович, д.т.н., проф., Кафедра "Автоматизированные системы управления", Южно-Российский государственный технический университет (Новочеркасский политехнический институт), Россия, Новочеркасск
7. Кравчик Вячеслав Георгиевич, к.т.н., доц., Кафедра "Энергетика и БЖД", Институт сферы обслуживания и предпринимательства (филиал) ДГТУ в г.Шахты, Россия, Шахты

Международный программный комитет конференции

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

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

Основные секции конференции

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

Ключевые даты конференции

01.01.2014 – 15.04.2014 – приём заявок на участие в конференции и докладов к публикации
16.04.2014 – 30.04.2014 – рецензирование и корректировка присланных докладов
01.05.2014 – 09.05.2014 – рассылка уведомлений о принятии докладов на конференцию и приём квитанций об оплате оргвзноса
10–12 мая 2014 – формирование сборника трудов и сертификатов
01.07.2014 – 15.07.2014 – размещение на сайте конференции электронного макета сборника материалов с присвоенными ISBN-кодами и именных сертификатов, подтверждающих участие в конференции
II – III квартал 2014 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

Отпечатано 01.06.2013

Верстка – **Галиаскаров Э.Г.**
Дизайн обложки – **Олейник П.П.**
Корректоры – **Лаптев В.В., Ермаков И.Е.**
Ответственный за выпуск – **Галиаскаров Э.Г.**
Подписано в печать **25.06.2013**. Формат А4
Бумага офсетная. Печать цифровая
Усл.печ.л. **12,4**
Тираж 300 экз
Заказ № 958

ISBN 978-5-9903782-4-7

9 785990 378247

ISBN 978-5-9903782-5-4

9 785990 378254