Математика программных систем

Mathematics of Program Systems

2013

Межвузовский сборник научных трудов

Выпуск 10

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Пермский государственный национальный
исследовательский университет»

Федеральное государственное автономное образовательное
учреждение высшего профессионального образования
«Национальный исследовательский университет
"Высшая школа экономики"»
(Пермский филиал)

Государственное образовательное учреждение высшего
профессионального образования
«Кубанский государственный университет»

МАТЕМАТИКА ПРОГРАММНЫХ СИСТЕМ

Межвузовский сборник научных трудов

Выпуск 10

Пермь 2013

УДК 004.4+004.8+004.9
ББК 32.973
М34

М34 Математика программных систем: межвуз. сб. науч. тр. / под ред. А.И. Микова и Л.Н. Лядовой; Перм. гос. нац. исслед. ун-т. – Пермь, 2013. – Вып. 10. – 170 с.: ил.

ISBN 978-5-7944-2495-9 (вып. 10)
ISBN 978-5-7944-1741-8

В сборник включены статьи, относящиеся к следующим направлениям научной работы: моделирование и технологии разработки информационных систем; информационный поиск, системы основанные на знаниях; методы разработки прикладных программ. Часть статей подготовлена при поддержке РФФИ (проект № 12-07-00763-а) и Научного фонда НИУ ВШЭ (проект № 13-09-0143), научно-исследовательского проекта КубГУ и проекта 12/19т. В издании представлены результаты, полученные при выполнении этих проектов в 2013 г. Кроме того, в сборник включены статьи, посвященные использованию информационных технологий в науке и образовании.

Материалы сборника могут представлять интерес для научных работников, специалистов в области программной инженерии, разработчиков информационных систем различного назначения, студентов и аспирантов, изучающих информационные технологии.

УДК 004.4+004.8+004.9
ББК 32.973

Печатается по решению редакционно-издательского совета Пермского государственного национального исследовательского университета

Рецензенты: доктор физико-математических наук, профессор, директор учебного центра «Информатика» ***С.В. Русаков***;
кафедра информационных технологий в бизнесе НИУ ВШЭ – Пермь

ISBN 978-5-7944-2495-9 (вып. 10)
ISBN 978-5-7944-1741-8

ПРЕДИСЛОВИЕ

Большая часть работ, включённых в настоящее издание межвузовского сборника научных статей, представляет результаты исследований, выполненных при финансовой поддержке РФФИ и программы софинансирования грантов РФФИ и РГНФ Научного фонда НИУ ВШЭ.

Первый раздел сборника содержит статьи, отражающие основные результаты, полученные в рамках реализации проекта № 12-07-00763-а «Методы и средства реализации трансформаций предметно-ориентированных языков моделирования сложных проблемно-ориентированных информационных систем», поддержанного грантом РФФИ, а также проекта № 13-09-0143, поддержанного Научным фондом НИУ ВШЭ. В представленных статьях описаны математические модели, лежащие в основе реализации DSM-платформ и CASE-средств. Приведен аналитический обзор средств трансформации моделей, разрабатываемых с использованием предметно-ориентированных языков, создаваемых при помощи современных языковых инструментариев.

В первый раздел также включены статьи, представляющие результаты, полученные при создании систем имитационного моделирования (работа выполнена при поддержке научно-исследовательского проекта КубГУ «Разработка и реализация информационно-аналитической среды, предназначенной для исследования и проектирования информационно-коммуникационных систем» и проекта 12/19т).

Отдельные статьи посвящены созданию систем интеллектуального анализа документов (Интернет-контента), систем компьютерного зрения.

Во второй раздел сборника включены статьи, в которых рассматриваются вопросы использования информационных технологий в образовании и науке.

МЕТОДЫ И СРЕДСТВА РАЗРАБОТКИ ПРОГРАММНЫХ СИСТЕМ

В.В. Ланин

Национальный исследовательский университет
«Высшая школа экономики»
(Пермский филиал)

vlanin@live.com

КЛАССИФИКАЦИЯ ФОРМАТОВ ЭЛЕКТРОННЫХ ДОКУМЕНТОВ

Введение

Вопрос выбора представления (формата) электронных документов является крайне важным. От представления документа зависят способы его обработки, спектр решаемых задач, для которых применим данный формат, и, в конечном итоге, удобство работы с ним пользователей. Особые возможности для обработки документов обеспечивает их семантическая индексация.

Для выбора нужного формата представления документа необходимо оценить все возможности, обеспечиваемые использованием этого формата. Для решения данной задачи полезно выполнить классификацию существующих форматов. При этом особое внимание следует уделить возможности включения метаданных как основного механизма поддержки семантического индексирования.

Цель данной работы – построить классификацию электронных документов, их форматов, которая может быть использована для выбора оптимального формата при решении задач представления и хранения, обработки электронных документов в информационных системах различного назначения.

Существующие классификации электронных документов

Рассмотрим понятие формата документа и связанное с ним понятие формата файла. Заметим, что хотя эти понятия используются повсеместно, но четкая их формулировка до сих пор не устоялась, исследователи продолжают дискуссию по формулировке данных определений [1-5].

Формат файла – набор семантических, синтаксических схем и правил сериализации для преобразования абстрактной информации в определенные наборы байт [4]. В свою очередь, *формат документа* – это формат файла для хранения документов на носителе информации.

В настоящее время используется большое количество различных форматов, существенно отличающихся по концепции и возможностям представления содержания. Общепринятой классификации форматов на данный момент нет. Чаще всего форматы документов классифицируют *в зависимости от задач*, на решение которых они ориентированы, и *подходов к обработке документов* в этих форматах. Например, в [6] вводится классификация, учитывающая особенность обработки документов при решении задач поиска. Форматы делятся на три класса:

1) устаревшие и закрытые коммерческие форматы (Microsoft Word, Lexicon, WordStar);
2) форматы, ориентированные на графическое представление документа (PostScript, PDF);
3) форматы, содержащие динамически выполняющиеся элементы (современные Интернет-сайты).

В [7] описана классификация электронных документов (ЭД) *по метаданным*, которые исследователь определяет как характеристики технологических процессов, необходимых для визуализации. Автор отмечает, что современные ЭД – это комбинированные сложноформатные документы, а тенденции, наблюдаемые в компьютерной отрасли, говорят об их дальнейшем усложнении. В основе классификационной модели – *ранжирование ЭД* в зависимости от комбинации метаданных. Выделяется пять классов документов:

- *Одноранговые* документы состоят из одного типа данных, объединенных одной структурой, записанных в один файл определенного формата (простые текстовые файлы, графические изображения, оцифрованные аудио- и видеодокументы).
- *Двухранговые* документы состоят из нескольких типов данных, объединенных одной структурой, записанных в один файл определенного формата (файлы электронных таблиц, документы векторных графических форматов).

– *Трехранговые ЭД* состоят из нескольких типов данных, объединенных в несколько структур, записанных в один файл определенного формата.
– *Четырехранговые ЭД* состоят из нескольких типов данных, объединенных в несколько структур, записанных в нескольких файлах общего формата.
– *Пятиранговые ЭД* состоят из нескольких типов данных, объединенных в несколько структур и записанных в нескольких файлах разных форматов.

Внутри ранга электронные документы можно разделить *по конкретным технологиям, в рамках которых они были созданы*: текстовым или графическим процессорам, типам СУБД, гипертекстовым системам и т.п.

Ранжирование ЭД позволяет подходить к ним с общих методологических позиций. Рассмотрение происходит в зависимости от объема материальных и интеллектуальных затрат, необходимых для проведения последующей миграции. Чем выше ранг ЭД, тем сложнее ее осуществить. Для обеспечения долговременной сохранности четырех- и пятиранговых ЭД требуются большие машинные ресурсы, труд высококвалифицированных программистов и ясное представление о их строении.

Следует упомянуть также *стандарт MIME* (Multipurpose Internet Mail Extensions – многоцелевые расширения Интернет-почты) – стандарт, описывающий передачу различных типов данных по электронной почте, а также, шире – спецификация для кодирования информации и форматирования сообщений таким образом, чтобы их можно было пересылать по сети Интернет.

Многоаспектная классификация форматов документов

Для анализа возможностей различных форматов ЭД введем *многоаспектную классификацию*.

В зависимости от *типа лицензии* на него формат может быть проприетарным или открытым. *Проприетарные* (*proprietary*) форматы создаются и контролируются частными организациями и являются собственностью авторов или правообладателей. *Открытый* (*non-proprietary*) формат обычно разрабатываемая некоммерческой организацией по стандартизации, спецификация формата доступна, использование формата свободно от лицензионных ограничений.

Спецификация формата может быть *закрытой* (*closed specifications*) или *открытой* (*open specifications*). Доступность специ-

фикации часто связывают с проприетарностью формата, что в большинстве случаев действительно так (формат Microsoft Word), но встречаются и исключения (формат PDF). Для поддержки форматов с закрытой спецификаций в продуктах третьих лиц применяется обратный инжиниринг, что зачастую негативно влияет на стабильность работы и нарушает условия использования формата.

В зависимости от *способа представления данных в файле* можно выделить *бинарные* и *текстовые форматы*. Первым форматом, используемым для представления текстовой информации, является *простой текстовый формат* (*plain text*), в котором в явном виде представлены символы текста, а из служебных символов используются только команды управления кареткой. Практически без каких-либо изменений он повсеместно применяется и по сей день. Одно из основных преимуществ текстового формата заключается в возможности его обработки без каких-либо специализированных программных средств. На базе текстового представления файла построено множество других форматов (например, RTF (*Rich text format*), XML и др.). Основными недостатками текстового представления являются значительный объем занимаемой памяти и последовательный доступ к содержимому. *Бинарные форматы*, напротив, достаточно компакты и поддерживают произвольный доступ к содержимому. Однако для доступа к их содержимому требуются знания об их структуре, что часто используется как «защищающий» фактор для проприетарных форматов.

Если классифицировать форматы электронных документов *по возможностям форматирования* (под форматированием понимается изменение «внешнего вида» текста, при котором не изменяется его содержание), то можно выделить следующие классы:

- «простые» текстовые форматы, ориентированные на представление только текстовой информации и не допускающие форматирования без изменения содержания;
- текстовые документы с поддержкой форматирования.

Также можно ввести разделение форматов электронных документов в зависимости *от способа использования*:

- форматы, ориентированные на редактирование;
- форматы, ориентированные на представление информации.

По *возможности расширения формата* (включения в файл дополнительных метаданных):

- расширение формата невозможно;
- расширение формата возможно неспецифичными средствами;
- расширение формата поддерживается спецификацией.

Результаты проведённой классификации представлены в табл. 1.

Таблица 1. Классификация наиболее популярных форматов ЭД

Формат	Тип лицензии	Спецификация	Способ представления	Форматирование	Назначение формата	Расширение формата
eXtensible Markup Language, XML	откр.	откр.	текст.	нет	ред.	есть
HyperText Markup Language, HTML	откр.	откр.	текст.	есть	просм.	нет
Rich Text Format, RTF	пропр.	откр.	текст.	есть	ред.	нет
Open XML	откр.	откр.	текст.	есть	ред.	есть
OpenDocument Format, ODF	откр.	откр.	текст.	есть	ред.	есть
Portable Document Format, PDF	пропр.	откр.	бинарн.	есть	просм.	нет
XML Paper Specification, XPS	откр.	откр.	текст.	есть	просм.	есть
Microsoft Word, Doc	пропр.	закр.	бинарн.	есть	ред.	есть
TeX	откр.	откр.	текст.	есть	просм.	нет
WordPerfect	пропр.	откр.	бинарн.	есть	ред.	нет
DjVu	пропр.	откр.	бинарн.	нет	просм.	нет

Заключение

Представленная классификация отражает все существенные характеристики форматов ЭД и может быть использована для выбора оптимального формата при решении задач представления и хранения, обработки документов в информационных системах различного назначения.

Библиографический список

1. *Brown A.* Selecting file formats for long-term preservation The National Archives (UK) Digital preservation guidance note 1. 2008. URL: http://www.nationalarchives.gov.uk/documents/selecting-file-formats.pdf.
2. *Авдеев А.Н., Ларин М.В.* Об организации электронного документооборота // Документация в информационном обществе: проблемы оптимизации документооборота: докл. и сообщ. на XVIII Междунар. науч.-практ. конф., 26-27 окт. 2011 г. / Федер. арх. агентство (Росархив), Всерос. науч.-исслед. ин-т документоведения и арх. дела (ВНИИДАД). М.: ВНИИДАД, 2012. С. 389-404.
3. *McLellan E.P.* General Study 11 Final Report: Selecting Digital File Formats for Long-Term Preservation. URL: http://www.interpares.org/display_file.cfm?doc=ip2_gs11_final_report_english.pdf.
4. *Todd M.* Technology Watch Report: File formats for preservation. URL: http://www.dpconline.org/component/docman/doc_download/375-file-formats-for-preservation.
5. *Abrams S.* Instalment on “File Formats”. URL: https://www.era.lib.ed.ac.uk/bitstream/handle/1842/3351/Abrams%20file-formats.pdf?sequence=1.
6. *Губин М.В.* Модели и методы представления текстового документа в системах информационного поиска: автореф. дис. канд. физ.-мат. наук, СПб., 2005.
7. *Тихонов В.И.* Сущностные характеристики, состав и классификация электронных документов // Документация в информационном обществе: электронное делопроизводство и электронный архив. М.: Росархив. ВНИИДАД. РОИА. 2000. С. 204-218.

В.В. Ланин, А.Б. Печенежский

Национальный исследовательский университет
«Высшая школа экономики»
(Пермский филиал)

vlanin@live.com

Пермский государственный национальный
исследовательский университет

nextzucker@gmail.com

ИНТЕЛЛЕКТУАЛЬНЫЙ СЕРВИС АНАЛИЗА ИНТЕРНЕТ-КОНТЕНТА НА ОСНОВЕ ОПИСАНИЯ ПРЕДМЕТНОЙ ОБЛАСТИ

Введение

В настоящее время Интернет является основным и наиболее богатым источником бизнес-информации и знаний. Задача извлечения и агрегации информации, представленной в Интернете, актуальна во многих областях: мониторинг СМИ и социальных сетей, анализ новостных лент. Изначально эта информация ориентирована на человека, следовательно, для её автоматической обработки необходимо использовать интеллектуальные методы. HTML-документы, являющиеся основным носителем информации в Интернет, слабоструктурированы, то есть имеют некоторую внутреннюю структуру, но допускают вхождение в структурный элемент неструктурированного текста. При анализе (структурном и содержательном) слабоструктурированных источников находят широкое применение методы Text Mining, а для Web-пространства можно говорить о Web Mining [7, 8].

Традиционно решение задачи извлечения данных является уникальным для каждого конкретного случая [3, 5]. В настоящей статье

описывается подход к автоматическому сбору структурированной информации из неструктурированных Интернет-документов, позволяющий унифицировать решение этой задачи при реализации проекта создания сервиса анализа Интернет-контента.

Постановка задачи

В рамках проекта необходимо реализовать *интеллектуальный сервис анализа Интернет-контента на основе описания предметной области*.

Сервис на входе должен получать от пользователя список адресов страниц, которые необходимо проанализировать, описание предметной области в виде онтологии, описание анализируемых ресурсов также в виде онтологии. Затем поисковый робот (Web-краулер), являющийся компонентом сервиса, должен выполнять обход каждого ресурса. После этого сервис должен определять информативную часть каждой страницы, извлекать информацию из выделенной информативной части и сохранять в структурированную базу данных (БД). Данный способ работы сервиса обеспечивает инвариантность относительно предметной области и, если необходимо проанализировать новые ресурсы, то никакого изменения программного кода не требуется – меняются описания предметной области и ресурсов. Тем самым сложность поиска остается неизменной.

Кроме того, сервис должен выполнять мониторинг ресурсов, то есть повторять анализ через определенные промежутки времени. При этом необходимо выделять повторно извлеченную информацию, чтобы не сохранять (не дублировать) её в БД.

Детализируем *требования к поисковому роботу*. На вход Web-краулера поступает список начальных URL-адресов; затем для каждого URL-адреса он выполняет загрузку страницы и передает загруженную страницу анализатору страниц. От анализатора страниц поисковый робот получает URL-адреса, извлеченные с загруженных страниц. Добавление новых URL-адресов завершается после достижения определенной глубины обхода.

Кроме того, Web-краулер должен:

- следить за повторяющимися страницами;
- учитывать возможные ошибки при загрузке страниц;
- как можно раньше выявлять страницы, которые не подходят для анализа;
- обрабатывать код HTML, в котором допущены ошибки;
- поддерживать динамическую загрузку страниц;

- переводить относительные URL в абсолютные;
- использовать канонический URL, чтобы избегать дублирования страниц;
- использовать эвристические правила для отбора страниц (ограничение длины адреса, наличие или отсутствие определенных слов и пр.);
- в целях повышения эффективности хранить несколько страниц в одном большом файле, используя XML-разметку;
- соблюдать «сетевой этикет» (идентифицировать себя, не отправлять слишком много запросов на один сервер, учитывать robots.txt).

Для каждой загруженной страницы анализатор страниц должен выполнять перечисленные ниже действия:

1. Построение DOM-дерева.
2. Извлечение URL-адресов.
3. Нахождение информативной части страницы.
4. Анализ информативной части.

При анализе информативной части анализатором страниц должен поддерживаться следующий *алгоритм работы*:

1. Разбиение текста на абзацы, предложения, слова.
2. Лемматизация слов и нормализация других объектов (числа, даты). При этом должна быть обеспечена возможность изменения словаря лемматизации [2].
3. Нахождение объектов поиска и определение их типов (например: ФИО, компания). На данном этапе должна быть возможность использовать справочные списки, содержащие имена собственные. Если слово находится в каком-либо списке, то оно принадлежит категории объектов, которую описывает этот список.
4. Установление связей между объектами. Для данной задачи предлагается использовать правила анализа, описанные в виде регулярных выражений, и строить деревья разбора.

Данные шаги должны выполняться на основе *онтологии предметной области* (формализованного описания предметной области). Для уменьшения пространства поиска при анализе страницы используется *онтология ресурсов* (описание ресурсов).

Таким образом, разработка процедуры анализа каждой страницы заменяется созданием *универсального поискового робота*, *анализатора страницы* и созданием *онтологий* для извлечения информации из текстов в новой предметной области [5, 6, 9]. Следовательно, единственным трудозатратным процессом является создание прикладной онтологии, требующее наличия эксперта.

Для реализации сервиса к создаваемым онтологиям предъявляются следующие требования:

1. Простота: онтология должна включать минимально возможное количество связей и понятий.

2. Возможность динамического изменения онтологии в процессе эксплуатации сервиса.

Для увеличения скорости анализа Интернет-ресурсов следует использовать описание Интернет-ресурсов, содержащее для каждого ресурса регулярное выражение для фильтра найденных на странице URL-адресов и XPath-запрос, позволяющий выделить в коде HTML место, где находится интересующий контент, и другую полезную информацию. Данное описание легко формируется при создании сервиса и при этом заметно увеличивает эффективность его работы, так как позволяет «игнорировать» ненужные URL-адреса и извлекать только полезную для пользователя информацию.

Обзор синтаксических анализаторов HTML

В процессе реализации программы потребуется выполнять синтаксический анализ HTML-страницы [10] для поиска нужной информации и ссылок на другие страницы. Для решения этой задачи необходимо выбрать библиотеку синтаксического разбора платформы .Net с открытым исходным кодом. Указанным требованиям удовлетворяют следующие библиотеки: HTML Agility Pack, SGMLReader.

HTML Agility Pack является свободной .Net библиотекой, позволяющей анализировать HTML-файлы. Доступны следующие возможности: Linq to Objects, XPath, XSLT. Методы библиотеки имеют названия, соответствующие интерфейсам DOM (Document Object Model Core), и позволяют получить доступ к содержимому HTML-страницы, представленному в виде дерева объектных узлов.

SGMLReader – C#-библиотека для преобразования кода HTML в XML. Способ работы этого анализатора аналогичен HTMLAgilityPack. HTML-документ преобразуется в DOM-структуру и позволяет работать с HTML-документом, как с XML. Доступны возможности Linq и XPath. SGMLReader, в отличие от HTMLAgilityPack, преобразует документ в объект типа XmlDocument, а его конкурент – в собственный тип HTMLDocument, являющийся оберткой XmlDocument. Благодаря чему программистам, имеющим опыт работы с XML-документами, привычней использовать SGMLReader. Следующим отличием является отсутствие встроенного загрузчика страниц.

Majestic-12 – это синтаксический анализатор .Net HTML с исходным кодом. Он создан для использования в распределенной поисковой

системе, в которой ежедневно требуется быстро обрабатывать терабайты HTML-документов, поэтому основное внимание при его разработке было уделено высокой производительности. Библиотека также отлично подходит для разбора XML, но не имеет Linq- и API-парсера.

Альтернативным способом анализа является перевод HTML-документа в xHTML-документ для возможности использования средства работы с XML-документами. Для получения xHTML-документа подойдет библиотека Tidy.Net, которая исправляет неверный код HTML и улучшает разметку.

Результаты проведённого аналитического обзора представлены в табл. 1.

Таблица 1. Сравнение синтаксических анализаторов

Требование	HTMLAgilityPack	SGMLReader	Majestic-12
LINQ	+	+	–
Производительность	Медленная обработка тегов	Медленная обработка тегов	Заточена на производительность
Встроенный загрузчик страниц	+	–	–
Документация и рабочие примеры в Интернете	Качественная документация и множество примеров	Плохая документация и сложность с нахождением примеров	Плохая документация и отсутствие примеров
Исправление ошибочной HTML-разметки	+	+	–

Обзор систем анализа Интернет-контента

Задача извлечения и агрегации информации актуальна во многих областях: например, мониторинг средств массовой информации, социальных сетей, новостных лент. Сервисы, решающие данную задачу, можно разделить по двум признакам: способ анализа и область поиска (табл. 2).

Одним из популярных сервисов является проект 80legs, который предоставляет пользователю доступ к легко настраиваемому Web-краулеру для сбора данных из Интернета. Пользователь вводит список URL-адресов, с которых необходимо начинать обход, указывает какую

информацию необходимо сохранять с помощью ключевых слов, регулярных выражений или собственного java-алгоритма. Возможности сервиса не позволяют учитывать семантику. Кроме того, 80legs даёт доступ к обновляемым пакетам с записями о различных исследованиях, которые пополняются на 10 миллионов записей каждый месяц. 80legs используется сотнями компаний для поиска рекламных каналов, для улучшения почтового спам-фильтра и пр.

В отдельную группу можно выделить системы мониторинга социальных медиа; например, такие проекты как youscan, iqbuzz, wobot. Они используют различные статистические алгоритмы для анализа Интернет-контента и представляют извлеченные данные в структурированном виде. Отметим, что данные проекты ограничивают область поиска.

Системы, которые используют методы анализа, учитывающие семантику, являются, как правило, исследовательскими, например Ontos [1], PULS. При этом в данных системах область поиска часто ограничена.

Таблица 2. Сравнение синтаксических анализаторов

Название	Способ анализа	Область поиска
80legs.com	Ключевые слова. Регулярные выражения	Без ограничения
YouScan.ru **Iqbuzz.ru** **Wobot.ru**	Статистические методы	Социальные сети. СМИ
ontos.com **PULS**	Семантические методы	Новостные ленты

Архитектура системы и подходы к реализации

Интеллектуальный сервис анализа Интернет-контента должен извлекать информацию из источников, расположенных в сети Интернет. При этом поиск и анализ HTML-документов и Web-ресурсов должен происходить автоматически. Для решения данной задачи разработана архитектура системы, представленная на рис. 1.

Сервис состоит из двух основных компонентов: *поискового робота* и *анализатора страниц*. Кроме того, сервис включает *механизм журнализации* и *менеджер настроек*. Компонент журнализации используется для записи информации о работе системы. Логирование используется как метод отладки программного обеспечения, а также способ мониторинга работы системы. Менеджер настроек позволяет

производить динамическую настройку системы. Пользователь взаимодействует с базой данных адресов, описанием предметной области и описанием Интернет-ресурса.

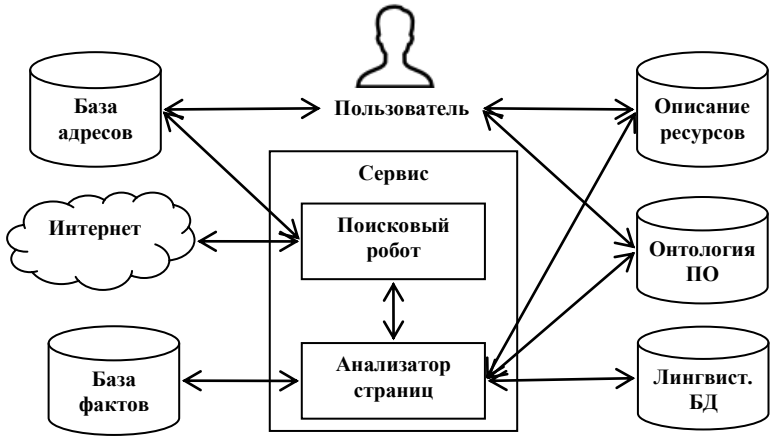


Рис. 1. Архитектура системы

База данных (БД) адресов хранится в формате XML, что позволяет хранить для каждого адреса дополнительную информацию, например глубину поиска или необходимость динамической подгрузки страницы. Описание предметной области и описание Интернет-ресурсов хранятся в формате OWL, а база данных фактов – в реляционной базе данных.

Благодаря использованию описания предметной области достигается инвариантность сервиса относительно предметной области. А описание Интернет-ресурсов позволяет увеличить эффективность анализа за счёт учёта знаний о ресурсах при обходе и извлечении информации.

Рассмотрим архитектуру поискового робота (рис. 2), который состоит из менеджера адресов и загрузчика. Алгоритм работы робота включает несколько этапов. Пользователь добавляет адреса ресурсов для анализа в базу данных адресов, менеджер адресов перебирает эти адреса. Менеджер адресов получает адрес из БД адресов и передает его загрузчику, который загружает страницу по данному адресу из сети Интернет, проверив возможные ошибки, и передаёт её анализатору страниц. Анализатор страниц кроме извлечения информации находит ссылки в HTML-документе и передает их менеджеру адресов, который

сохраняет новые адреса в БД адресов.

База данных адресов хранит информацию о посещенных страницах и страницах, которые ожидают анализа. Благодаря чему исключается повторный анализ страниц, расположенных по одинаковым адресам в текущую сессию загрузки.

Рис. 2. Архитектура поискового робота

Рассмотрим архитектуру анализатора страниц (рис. 3). Основными компонентами здесь являются анализатор HTML-документа, анализатор текста, менеджер описаний.

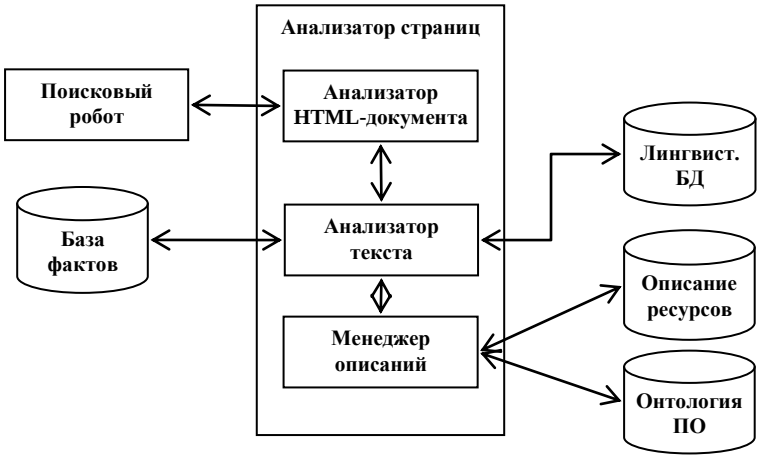


Рис. 3. Архитектура анализатора страниц

Поисковый робот передает загруженную страницу анализатору HTML-документа, который строит DOM-дерево HTML-документа, извлекает информативную часть страницы и передает её анализатору текстовой информации.

Анализатор текста, используя базу лемматизатора и описание предметной области, через менеджер описаний извлекает факты предметной области и сохраняет их в БД фактов. При этом для получения знаний о текущем Интернет-ресурсе анализатор страниц использует его описание. В качестве редактора описаний предполагается использовать Protégé.

База данных фактов предназначена для хранения извлеченной информации. При сохранении нового факта необходима проверка на наличие аналогичного объекта в базе фактов. Кроме того, в базе данных хранится информация о пройденных страницах. Описание предметной области создаётся заранее и содержит термины предметной области, связи между ними.

Заключение

Прораммы представленной системы разрабатываются в Microsoft Visual Studio, а в качестве СУБД используется Microsoft SQL Server. Для реализации компонента разбора HTML выбрана библиотека Agility Pack. Для лемматизации используется библиотека морфологического анализа на основе словаря А.А. Зализняка (MCR.DLL). А для работы с описанием предметной области, которое представляется в виде онтологии и хранится в формате OWL, используется библиотека OWLdotnetapi.

Преимущества разрабатываемой системы обеспечиваются применением описаний предметной области и ресурсов при анализе Интернет-контента. Это позволяет извлекать более качественную информацию и даёт возможность анализировать данные без ограничения области поиска. Кроме того, для расширения множества анализируемых источников не требуется вносить изменения в код программ, следовательно, сложность разработки не изменяется.

Библиографический список

1. *Хорошевский В.Ф.* OntosMiner: семейство систем извлечения информации из мультиязычных коллекций документов // Труды IX Национальной конференции по искусственному интеллекту с международным участием КИИ-2004. В 3 т. М.: Физматлит, 2004. Т. 2. С. 573-581.

2. *Лукашевич Н.В, Добров Б.В.* Отношения в онтологиях для решения задач информационного поиска в больших разнородных текстовых коллекциях // Труды IX Национальной конференции по искусственному интеллекту с международным участием КИИ-2004. В 3 т. М.: Физматлит, 2004. М.: Физматлит, 2004. Т. 2. С. 544-551.
3. *Chugunov A., Lanin V.* Intelligent search based on ontological resources and graph models // Proceedings of the 7th Spring/Summer Young Researchers' Colloquium on Software Engineering, SYRCoSE 2013 / Ed. by: A. Petrenko; A. Terekhov; A. Kamkin. Kazan, 2013. P. 133-135.
4. *Lanin V., Nesterov R., Osotova T.* Intelligent Service for Aggregation of Real Estate Market Offers // Proceedings of the 7th Spring/Summer Young Researchers' Colloquium on Software Engineering, SYRCoSE 2013 / Ed. By: A. Petrenko; A. Terekhov; A. Kamkin. Kazan, 2013. P. 136-138.
5. *Buitelaar P., Cimiano P., Frank A., Hartung M., Racioppa S.* Ontology-based information extraction and integration from heterogeneous data sources. Int. J. Hum.-Comput. Stud. 66, 11. 2008. P.759-788.
6. *Grigalis T.* Towards web-scale structured web data extraction // Proceedings of the sixth ACM international conference on Web search and data mining (WSDM '13). ACM, New York, NY, USA, 2013. P. 753-758.
7. *Stumme G., Hotho A., Berendt B.* Semantic Web Mining // Web Semant. № 4, V.2. 2006. P. 124-143.
8. *Liu B.* Web Data Mining. Exploring Hyperlinks, Contents, and Usage Data. Chicago:Springer, 2007. 532 p.
9. *Weal M.J., Kim S., Lewis P.H., Millard D.E., Sinclair P.A.S., De Roure D.C., Nigel R.* Ontologies as facilitators for repurposing web documents / Shadbolt. Southampton, 2007. P. 537-562.
10. *Xu Z., Yan D.* Designing and Implementing of the Webpage Information Extracting Model Based on Tags // Proceedings of the 2011 International Conference on Intelligence Science and Information Engineering (ISIE '11). IEEE Computer Society, Washington, DC, USA, 2011. P. 273-275.

О.Н. Лапина

Кубанский государственный университет

olgaln09@gmail.com

МЕТОДЫ И АЛГОРИТМЫ ТРАНСЛЯЦИИ ОПИСАНИЙ Р-ГРАФОВ[1]

Введение

Р-графы – это особый вид графовых структур, которые используются для моделирования различных систем. Объектами моделирования могут являться вычислительные системы и их компоненты (процессоры, элементы памяти, цифровые электронные схемы, соединения между ними), а также компьютерные сети, их узлы и линии связи (проводные и беспроводные).

Цель данной работы – представление проекта разработки компонента трансляции описаний структур, описанных на основе *Р*-графов.

Основные понятия

Моделируемые объекты состоят из элементов (блоков, узлов) и связей между элементами. Вершина графа представляет собой элемент, а ребро или дуга графа – связь (проводник, шина). Так как связей между блоками может быть много, у вершин необходимо выделять входы и выходы (полюсы), т.е. соединяются не непосредственно вершина с вершиной, а полюс одной вершины с полюсом другой (соответствующие выходные и входные полюсы). Таким образом, для описания модели, её элементов и связей между ними получаем новые структуры – *Р-графы*. Сам *Р*-граф, как и его вершины, имеет полюсы (π).

[1] Работа выполнена при поддержке научно-исследовательского проекта КубГУ «Разработка и реализация информационно-аналитической среды, предназначенной для исследования и проектирования информационно-коммуникационных систем» и проекта 12/19т.

Таким образом, *P-граф* может быть представлен как тройка $P = \{U, V, W\}$, где V – множество вершин графа с внутренними полюсами, W – набор дуг, связывающих вершины графа, U – набор внешних полюсов. Внутренние полюсы разделяют на входные $In(V)$ и выходные $Out(V)$.

Полюсы графа с помощью ребер (дуг) и вспомогательных элементов могут соединяться с другими графами. Такой граф в дальнейшем можно рассматривать как вершину, «расшифрованную» структурой, таким образом, возможно получение «масштабируемой» структуры, имеющей иерархическое представление.

Описание графовых структур в данной работе осуществляется средствами *языка моделирования Triad* [4].

Языковые средства описания структур

Язык моделирования Triad позволяет создать имитационную модель, представляющую собой совокупность объектов, которые действуют по определенным сценариям и обмениваются информацией друг с другом.

Модель может быть представлена тройкой $\mu = \{Str, Rout, Mes\}$ – слоями структур, рутин и сообщений. *Слой структур* (*Str*) предназначен для описания моделируемых объектов и связей между ними. *Слой рутин* (*Rout*) представляет собой набор алгоритмов для описания поведения моделируемых объектов. *Слой сообщений* (*Mes*) дает возможность описывать сообщения сложной структуры.

Имитационная модель в Triad является иерархической. Каждый уровень модели можно описать как *P*-граф. Внутренние полюсы графа используются для передачи сообщений на одном уровне иерархии. Набор внешних полюсов служит для передачи информации между объектами, находящимися на различных (смежных) уровнях иерархии.

Основой описания слоя являются *структурные константы*. Имеется три типа структурных констант: полюсы, вершины и графы.

Полюс – элементарная структурная единица. Полюс-константа записывается в виде *polus*(<простое имя>), где <простое имя> – это идентификатор или идентификатор со списком индексов.

Вершина-константа записывается в виде *node*(<простое имя>). Поскольку вершина определяется как множество полюсов, то в записи вершины-константы могут присутствовать имена её полюсов, задаваемые перечислением: *node*(<простое имя> (<список простых имен>)). Имена в списке перечисляются в произвольном порядке. Примеры: *node*(*A*), *node*(*B*[2]) – вершины, имеющие по одному полюсу, *node*(*C*(*P, Q*)) – вершина *C* с двумя полюсами *P* и *Q*.

Имя дуги состоит из имен инцидентных ей полюсов или вершин и знака направления дуги – двулитерного символа языка `–>` или `<–`. Для обозначения ребра вместо знака направления используется двулитерный символ `<>`.

Операция *добавления* вершины, дуги или ребра к графу обозначается символом `+`, а *удаления* – символом `–`. Эти же символы применяются для операции *объединения графов* (добавления к одному графу другого графа) и *удаления подграфа* графа.

Задать граф можно, перечислив все его вершины и дуги или указав все списки смежности, однако, основным способом является использование графовых констант, т.е. графов некоторого стандартного вида, что снижает трудоёмкость описания.

В набор *графов-констант* входят:

- простая цепь *path*(*n*), где *n* – количество вершин;
- простой цикл *cycle*(*n*), где *n* – количество вершин;
- полный *compl*(*n*), где *n* – количество вершин;
- решетка *rectan*(*m, n*), имеющая $n \times m$ вершин;
- решетка *rectan*(*k, m, n*), включающая $n \times m \times k$ вершин;
- дерево *tree*(*n, m*), где количество вершин – $(n^{m+1} - 1)/(n - 1)$;
- двудольный граф *bipart*(*n, m*), где количество вершин – $n+m$;
- звезда *star*(*n*), где общее количество вершин равно $n+1$.
- вполне несвязный *nc*(*n*), где *n* – количество вершин.

В общем случае структура (или *P*-граф) задается описанием вида

structure <наименование структуры> ***def*** (<тело описания>) ***endstr***

Тело описания может иметь различную степень сложности. Простейшее тело – оператор присваивания, в левой части которого находится имя структуры, а в правой – графовая константа, например:

structure A ***def*** A := ***path***(4)(M, N, K, L) ***endstr***

В случае, если одним оператором задать структуру трудно, тело может содержать последовательность операторов, задающую алгоритм *вычисления* структуры (*P*-графа). В описании могут присутствовать операторы, задающие вычисление структуры, имена графовых констант, знаки операторов, числа, скобки, имена вершин графа и т.п.:

```
structure Torus def
    Torus := rectan(4, 4)(P[1:4, 1:4](A, B, C, D));
    for i := 1 to 4 do
        Torus := Torus + (P[1,i](B) <> P[4,i](D))
                       + (P[i,1](A) <> P[i,4](C))
    endf;
endstr.
```

На входе программа-транслятор получает текстовый файл, содержащий описание структуры (*P*-графа) на языке Triad.

Входной файл может содержать ошибки. Задачей программного модуля является проведение синтаксического анализа, семантического анализа и генерации кода для описанных конструкций.

На этапе *лексического анализа* входной поток символов считывается и разбивается на лексемы – последовательности символов с определенным совокупным значением (идентификатор, число, символ и т.п.). Например, лексический анализатор строки *S* := *path*(2)(*a*, *b*) выведет поток лексем:

идентификатор *S*
:=
идентификатор *path*
(
2
)
(
идентификатор *a*
идентификатор *b*
)

На этапе *синтаксического анализа* выполняется группирование лексем в грамматические фразы.

Синтаксис описания структуры (*P*-графа) задается с использованием рекурсии с помощью расширенной БНФ:

<тело описания структуры> ::=
<описание переменных> <операторы построения структуры>

<операторы построения структуры> ::=
{<оператор построения структуры>**;**}

<оператор построения структуры> ::=
<оператор присваивания> | <оператор цикла> |
<условный оператор> |
<оператор задания полюса структуры>

<оператор присваивания> ::= <имя> **:=** <выражение>

<оператор цикла> ::=
for <идентификатор> **:=** <целочисленное выражение>
to <целочисленное выражение>
do <оператор построения структуры> ***endf***

<условный оператор> ::=

if <выражение>
then <оператор построения структуры>
else <оператор построения структуры>
endi |

if <выражение> ***then*** <оператор построения структуры>
endi

<список имен вершин> ::=
<имя группы вершин> {**,**<имя группы вершин>}

<имя группы вершин> ::=
<идентификатор>
[**[**<диапазон индексов>{**,**<диапазон индексов>}**]**]
[**(**<список имен полюсов>**)**]

<обращение к функции> ::=
<идентификатор> **(**<выражение>{**,**<выражение>}**)**

<имя> ::= <идентификатор> [**[**<индекс> {**,**<индекс>}**]**]
[**(**<идентификатор> **[**<индекс> {**,**<индекс>}**]** **)**]

<индекс> ::= <целочисленное выражение>

Для синтаксического анализа каждой конструкции реализована отдельная процедура, алгорим работы которой определяется соответствующим правилом. Каждая ошибка, обнаруженная в результате работы процедур синтаксического анализа, имеет свой тип. При обнаружении ошибок программа укажет, какая именно ошибка и где получена. В случае отсутствия ошибок транслятор формирует таблицу идентификаторов, содержащую параметры введенной структуры, и генерирует объектный код для её создания («вычисления») при выполнении программы моделирования.

Ход трансляции отображается на экране, результаты могут быть выведены в файл.

Рассмотрм пример синтаксического анализа строки

if M < N ***then*** …

В ходе выполнения процедура синтаксического анализа будут выведены следующие сообщения:

'if ' Начинается оператор.
Начинается условный оператор.

(т.е. синтаксический анализатор распознал и начал разбор условного оператора и далее происходит анализ в соответствии с правилами, описанными выше).

Пусть текст содержит ошибку в самом операторе:

ifa M < N ***then*** …

Процедура синтаксического анализа обнаружит ошибку и выведет соответствующее сообщение:

'ifa ' Начинается оператор.
AssignationOperator Begining
VariableName Begining
Current Variable is 'ifa'
'M' '<' IndexesList Begining
IndexesList Ended
VariableName Ended
Заканчивается оператор.
Заканчивается тело структуры.
Синтаксическая ошибка.

Если описание графа не содержит ошибок, генерируется код на языке C. Например, текстовый файл содержит описание некого *P*-графа без ошибок:

```
integer  m, i, n, nk;
for  i:=2  to  nk  do
  if m<n then Torus := Torus + (P[i,1](A) <– P[1,i](B)) endi
endf
```

В результате генерации получаем код на языке C:

```
int m, i, n, nk;
for  (int i = 2 ; i <= nk ; i++)
{
  if (m < n )
  { add_edge(Torus, // Имя структуры
            strcat(strcat((char*) malloc(sizeof(char)*10),"P"),
            strcat(itoa(i), strcat(itoa(1 ),""))), // Имя вершины
            strcat(strcat((char *) malloc(sizeof(char)*10),"A"), ""),
                        // Имя полюса
            strcat(strcat((char *) malloc(sizeof(char)*10),"P"),
            strcat(itoa(1), strcat(itoa(i ),""))), // Имя вершины
            strcat(strcat((char *) malloc(sizeof(char)*10),"B"), "")
                        // Имя полюса
            );
  }
}
return Torus;
```

Разработан также модуль, который генерирует программный код на языке Assembler. Данный модуль имеет ряд преимуществ, связанных с выбранным языком программирования. К ним можно отнести быстродействие ассемблерных кодов, чёткий контроль выделяемой памяти.

Программа-генератор получает на входе файл, содержащий дерево разбора строки, написанной на языке Triad.

Например, имеется строка *Path*(3)(*a*,*b*,*c*(*x*,*y*),*d*). Программа анализирует полученные данные. Результатом полного разбора приведённой строки будет следующая строка в памяти: 1*Path*|34|4*a*|4*b*|4*c*|6*x*|6*y*|4*d*|.

Из полученной «заготовки» последовательно извлекаются вершины, ребра, определяется тип графа по ключевому слову с кодом 1. Все эти данные записываются и используются для подстановки в шаблоны при генерации кода на языке Assembler. Полученный код записывается в файл типа .asm.

Вся информация о полученном графе помещается в сегмент данных. При этом все идентификаторы, которые использовались для названия вершин, ребер и т.п. заменяются своими порядковыми номерами, т.е. вместо имен вершин *a*, *b*, *c* мы получим коды V_1, V_2, V_3, где цифры соответствуют именам вершин.

После записи всех элементов графа и кодирования имён, создается таблица смежности графа следующего формата:

```
G_1_Def  dw   G_1_Size
         dw   4          ;число вершин
         dw   1          ;код вершины a
         dw   1          ;число исходящих связей
         dw   1,2,1      ;код полюса,
                         ;код вершины назначения,
                         ;код ее полюса
         dw   2          ;код вершины b
```

Таким образом, в работе используется метод трансляции – JIT, т.е. результат трансляции (код и таблицы) остается в оперативной памяти компьютера для последующего исполнения. Исполнение созданного кода приводит к построению в оперативной памяти внутреннего спискового представления структуры *P*-графа в виде совокупности связанных дескрипторов. Полученное представление *P*-графа можно использовать для вывода графа на экран либо для выполнения каких-либо операций по добавлению или удалению вершин, ребер или других графов для динамического «вычисления» нужной структуры в процессе преобразования модели при выполнении программы моделирования.

Заключение

В ходе работы над проектом создан программный модуль, представляющий собой транслятор описаний структур (*P*-графов), заданных на языке Triad. В случае отсутствия ошибок в описании структуры транслятор генерирует объектный код программы на языках C либо Assembler. Исполнение созданного кода приводит к построению в оперативной памяти внутреннего спискового представления структуры *P*-графа в виде совокупности связанных дескрипторов.

Полученное представление *P*-графа используется для создания симулятора – программы, которая моделирует работу распределенной системы. Симулятор позволит исследовать различные распределенные алгоритмы и различные конфигурации моделируемой сети.

Разработан также визуальный пользовательский интерфейс для системы моделирования в среде разработки QtCreator, позволяющий работать с входными и выходными файлами, редактировать их в панели текстового редактора, запускать процесс трансляции и моделирования, наблюдать за результатами работы в отладочной панели.

Реализованные средства могут быть оптимизированы и расширены в ходе дальнейших разработок.

Библиографический список

1. *Ахо А., Ульман Д.* Теория синтаксического анализа, перевода и компиляции. Т.1. Синтаксический анализ. М.: Мир, 1978.
2. *Ахо А., Ульман Д.* Теория синтаксического анализа, перевода и компиляции. Т.2. Компиляция. М.: Мир, 1978.
3. *Вирт Н.* Построение компиляторов. М.: ДМК Пресс, 2010.
4. *Mikov A.I.* Simulation and Design of Hardware and Software with Triad // Proceedings of, 2nd Intl. Conf. On Electronic Hardware Description Languages, Las Vegas, USA, 1995. P. 15-20.

Н.В. Лукиных, Л.Н. Лядова

Национальный исследовательский университет
«Высшая школа экономики»
(Пермский филиал)

LLyadova@hse.ru

О ПОДХОДЕ К РЕАЛИЗАЦИИ СРЕДСТВ ВЕДЕНИЯ НОРМАТИВНО-СПРАВОЧНОЙ ИНФОРМАЦИИ

Введение

Деятельность любого предприятия, вне зависимости от отраслевой специфики, связана с постоянным обращением к нормативно-справочной информации (НСИ). По мере роста и развития бизнес-системы количество такой информации быстро увеличивается. Данная информация поступает как из внешних источников (ГОСТы, справочники и классификаторы отраслевых министерств, ведомств), так и порождается внутри подразделений самой компании (бизнес-системы). У крупной или средней компании, в состав которой входит несколько предприятий, филиалов, подразделений, неизбежно возникает потребность в интеграции данных, которые поступают из разных источников [1, 3]. Эта задача может быть решена только на основе использования единых классификаторов, справочников и стандартизации способов представления и кодирования данных во взаимодействующих системах, обменивающихся информацией [13].

В соответствии с требованиями, предъявляемыми к проектам современных интегрированных автоматизированных информационных систем (ИС), необходимо обеспечить *однократность ввода данных* в рамках отрасли и *возможность интеграции* ИС различных уровней, принадлежащих различным учреждениям и организациям (бизнес-системам), в единую ИС отрасли. Одним из элементов, определяющих свойства системы, является реализация в ней установленных для соот-

ветствующей отрасли стандартов представления и кодирования информации [2, 3].

Универсальную систему управления НСИ следует рассматривать как совокупность методик ведения и поддержки НСИ, а также ряда технологических решений, обеспечивающих решение задач создания единого информационного пространства предприятия. Посредством такой системы должны быть обеспечены не только возможность интеграции систем на основе централизованного ведения НСИ, но могут также быть реализованы средства анализа данных, полученных из различных источников.

Средства управления НСИ должны не только обеспечивать работу программных компонентов ИС при выполнении пользователями операций автоматизируемых бизнес-процессов, но и предоставлять в распоряжение пользователей удобный пользовательский интерфейс, позволяющий создавать и модифицировать ресурсы, быстро искать необходимую информацию в соответствии с заданными критериями. При этом должна быть обеспечена возможность работы с классификаторами, имеющими различную структуру, использующими различные алгоритмы обработки информации, импорта классификаторов из внешних источников.

Средства управления нормативно-справочной информацией CASE-системы METAS предназначены для стандартизации представления и кодирования информации в информационных системах, унификации средств работы с нормативно-справочной информацией, в частности, с классификаторами различных видов. В основе реализации этих средств – разработанная авторами математическая модель [6, 16].

Описываемый комплекс программ предназначен в первую очередь для ведения НСИ в информационных системах, созданных на основе CASE-технологии METAS [8-10], для применения их в различных предметных областях без каких бы то ни было ограничений.

Программные компоненты могут быть также интегрированы и в информационные системы сторонних разработчиков для ведения классификаторов различных типов (иерархических, фасетных и пр.).

Понятие классификатора и задача управления НСИ

Классификатор – это документ, с помощью которого осуществляется формализованное описание информации в информационных системах. Классификатор содержит наименования объектов, наименования классификационных группировок и их кодовые обозначения.

Основными *объектами классификации и кодирования* являются

реквизиты-признаки (справочные и группировочные). Они описывают процессы, место, время выполнения процессов, субъекты и объекты действия, отражаемые в различных показателях, учитываемых и анализируемых в ИС. К объектам классификации и кодирования относятся также *наименования показателей* и *документов*, используемых в ИС.

Для кодирования объектов их необходимо упорядочить по некоторым признакам. Результат упорядоченного распределения объектов заданного множества и называется *классификацией*, а совокупность правил распределения объектов множества на подмножества – *системой классификации*. То свойство (или характеристика, атрибут) объекта классификации, которое позволяет установить его сходство или различие с другими объектами, называется *признаком классификации*. *Основанием классификации* называется тот признак, по которому ведется разбиение множества на подмножества на определенной ступени классификации. *Ступень классификации* – это результат очередного распределения объектов одной классификационной группировки. *Уровень классификации* – это совокупность классификационных группировок, расположенных на одних и тех же ступенях классификации. *Глубина системы классификации* – это количество уровней классификации, допустимое в данной системе.

Системы классификации являются основой для построения всех словарей и справочников ИС, обеспечивая единообразие представления информации, сопоставимость показателей при обработке и анализе данных.

Модель классификатора

В основе работы программных компонентов лежит математическая модель классификатора, разработанная авторами [6, 16]. Данная модель должна быть *универсальной* для различных типов классификаторов.

Под *универсальностью математической модели* будем понимать абстракцию, позволяющую описать:

- классификатор любого типа, формализуя общепринятые определения в области классификации и кодирования информации;
- представление классификатора в рамках ИС (на физическом, логическом и презентационном (визуальном) уровнях);
- средства управления классификаторами (создания, поддержки актуального состояния);
- алгоритмы преобразования к универсальной структуре классификаторов, импорта/экспорта классификаторов, приведение

импортируемых классификаторов в соответствие с единой системой управления.

Выбор в пользу *теоретико-графовой модели* аргументируется следующими фактами:

- Внутренняя структура классификатора изначально представляет собой граф, где вершины – это объекты классификации, а дуги – информация о соподчиненности объектов.
- Любая информационная система хранит данные в базе данных (БД). Структуру БД легко можно представить в виде графа, тогда представление классификатора в ИС сведется к отображению одного графа на другой.
- Пользователям удобно воспринимать классификатор в виде иерархии объектов – дерева. Дерево – это построение иерархии на множестве объектов классификации и отношений между ними. При этом часто деревом описываемая структура является только на экране пользователя (отсюда и название). С точки зрения структуры это ориентированный граф, возможно, с циклами.

В описании модели [6, 16] последовательно определяются все ее элементы – объекты классификации, классификаторы, отношения соподчиненности, операции над объектами классификации, отношениями соподчинённости и классификаторами.

Пусть имеется конечное линейно упорядоченное множество A предметных переменных $a_1, \ldots, a_n$, называемых *атрибутами*, с областями возможных значений $D_1, \ldots, D_n$ соответственно, называемыми *их доменами* и содержащими символ Λ – *неопределённое значение*.

Домены различных атрибутов могут совпадать. В этом случае атрибуты называются *однотипными*. Для любого подмножества $M \subseteq A$ декартово произведение D_M доменов всех атрибутов в M называется *доменом подмножества M*, а его элементы – *значениями* этого *подмножества*.

Объект классификации O_i – элемент классифицируемого множества, характеризующийся своими атрибутами $O_i = f(A_i) = A'$, где f – характеристическая функция, определяющая множество A' из A.

Структура классификатора, как правило, должна иметь три блока:

- *блок идентификации*, включающий коды объектов классификации и классификационных группировок;

– *блок наименований объектов и классификационных группировок* на естественном языке;
– *блок дополнительных признаков объектов*, включающий наименования и коды дополнительных признаков объектов классификации.

В соответствии со структурой классификатора *разделим атрибуты на три группы*:

– A_{iden} – *идентификационные атрибуты,* подмножество элементов из множества A (подмножество множества A_{iden} или все множество A_{iden} однозначно определяет объект классификации O_i):

$$A_{iden} \subseteq A^{'},\ A^{*}{}_{iden} \subseteq A^{'}, \exists! h = H(O_i(A^{*}{}_{iden}))$$

– A_{name} – *атрибуты наименования,* подмножество элементов из множества A;

– A_{add} – *дополнительные атрибуты,* подмножество элементов из множества A.

Признак классификации p_{ij} – свойство или характеристика объекта, по которому производится классификация, т.е. существует такая характеристическая функция f_i, аргументом которой является объект классификации, а результатом – значение признака классификации. Признак классификации иногда называют основанием деления:

$$\exists n \in N,\ j = \overline{1,n}, \forall p_{ij} = f_j\left[O_i(A_{iden})\right], P_i = \left\{p_{i1},...,p_{in}\right\}.$$

У объекта классификации может быть несколько признаков классификации. P_i – множество значений признаков классификации объекта O_i.

Классификационная группировка G_i – подмножество объектов, полученное в результате классификации. Это подмножество обладает совокупностью одних и тех же значений признаков классификации для каждого объекта классификации, и называется *множеством признаков группировки* G_i. Обозначим его P_i^G.

Классификационная группировка имеет *код* и *наименование* в системе классификации и может представлять собой объект классификации в классификаторе:

$$G_i = \left\{O_{i1},...O_{in}\right\}, \forall j,k \subseteq n : P_{ij} = P_{ik} = P_i^G.$$

Всегда существует некоторое *правило*, по которому определяется, принадлежит ли объект классификации данной классификационной группировке.

В качестве правила будем использовать предикат $Rule_{G,O}(A_{iden})$. Если значение предиката – истина, то объект O принадлежит группировке G, если ложно, то нет:

$$Rule_{G,O}(A_{iden}) = \text{true} \Leftrightarrow (\exists A^{O}_{iden} \in A'): f_{rule}(A^{O}_{iden}) = f_{rule}(A^{G}_{iden})).$$

Как уже говорилось выше, классификационные группировки могут отражаться или не отражаться в кодах классификаторов. Например, в классификаторе ОКАТО (общероссийский классификатор административно-территориальных образований), в коде отражается группировка, то есть любой код ОКАТО несет смысловую нагрузку, некоторые позиции символов в коде отвечают за уровень группировки. Группировки могут задаваться независимо от кода, например в ОКФС (общероссийский классификатор форм собственности) используется сквозная нумерация кода, а группировки задаются специальным алгоритмом.

В результате объект классификации можно представить как тройку:

$$O_i = (A', P_i, G_i),$$

где A' – множество атрибутов объекта O_i;

P_i – множество признаков классификации объекта O_i;

G_i – классификационная группировка, к которой относится объект классификации O_i.

Рассмотрим пример представления объектов классификации в соответствии с описанными определениями. Пусть есть текстовое описание классификации (пример взят из ОКАТО):

Классификатор включает объекты административно-территориального деления, кроме сельских населенных пунктов, которые в настоящее типографское издание не входят из-за большого объёма этой информации.

Каждая позиция классификатора структурно состоит из 3-х блоков:

1) блок идентификации объекта;
2) блок наименования объекта;
3) блок дополнительных данных.

Блок идентификации объекта включает идентификационный код и контрольное число. Идентификационный код строится с использованием серийно-порядкового, последовательного и параллельного методов кодирования.

Длина кода – от 2-х до 8-ми разрядов в зависимости от уровня классификации, на котором находится объект.

Структура кодового обозначения в блоке идентификации:

XX XXX XXX КЧ,

где

1,2 знаки – объекты первого уровня классификации;
3,4,5 знаки – объекты второго уровня классификации;
6,7,8 знаки – объекты третьего уровня классификации;
КЧ – контрольное число.

Контрольное число в кодовых обозначениях рассчитывается по действующей в ЕСКК методике расчета и применения контрольных чисел.

Для кодирования большинства объектов ОКАТО используется следующая структура кода. Разряды 1 и 2 предназначены для кодирования объектов федерального значения, расположенных на первом уровне классификации. Система кодирования этих объектов – серийно-порядковая, обеспечивающая преемственность с ранее действующим классификатором СОАТО. Разряды 3, 4, 5 используются для кодирования объектов второго уровня классификации, разряды 6, 7, 8 – для кодирования объектов третьего уровня. В этих случаях применяется последовательный метод кодирования. При этом разряды 3 и 6 отведены под признаки соответственно Р1 и Р2, указывающие уровень классификации и вид кодируемого объекта. В этом случае применяется параллельный метод кодирования.

Признак второго уровня классификации – Р1 (разряд 3) имеет значение:

1 – автономный округ;
2 – район (в том числе внутригородской), округ;
4 – город, посёлок городского типа.

Признак третьего уровня классификации – Р2 (разряд 6) имеет значение:

3 – внутригородской район, округ города;
5 – город, посёлок городского типа;
8 – сельсовет.

Из описанного выше текста можно выделить 5 признаков классификации. Запишем признаки в формате

«знак в блоке идентификации» = «какие значения может принимать»:

$p_1 : A_{iden}[1,2] = [0..9][1..9]$ *принадлежность* 1 *уровню вложенности*
$p_2 : p_1 \,\&\, A_{iden}[3,4,5] = [1..9][1..9][0..9]$ *принадлежность* 2 *уровню вложенности*
$p_3 : p_2 \,\&\, A_{iden}[6,7,8] = [1..9][1..9][0..9]$ *принадлежность* 3 *уровню вложенности*
$p_4 : A_{iden}[3] = [1,2,4]$ *соответствие признаку* $P1$
$p_5 : A_{iden}[6] = [3,5,8]$ *соответствие признаку* $P2$

Таким образом, в классификационную группировку первого уровня G_1 попадают все объекты классификации O, у которых выполняется признак классификации p_1 (первые две позиции в блоке идентификации – цифры) и остальные позиции – нули:

$$G_1 = \{O_1 ... O_n\}, P_{G1} = \{\forall i = \overline{0,9}, \forall j = \overline{1,9} : p_1(i, j) \,\&\, A_{iden}[3-8] = ij000000\}.$$

Аналогичного для группировок G_{xy} второго уровня вложенности должен выполняться признак p_1, p_2 и остальные позиции – нули:

$$G_{xy} = \{O_{11} ... O_{nn}\},$$

$$P_{Gxy} = \{\forall i = \overline{1,9}, \forall j = \overline{1,9}, \forall k = \overline{0,9} : p_1(x, y) \,\&\, p_2(i, j, k) \,\&\, A_{iden}[678] = xyijk\,000\}$$

Классификатор – это документ, с помощью которого осуществляется формализованное описание информации в информационных системах, содержащий наименования объектов, наименования классификационных группировок и их кодовые обозначения.

Пару $K = (V, E)$, где V – некоторое множество классификационных объектов с различными атрибутами и E – отношение соподчиненности на нем, назовём *классификатором.*

Введем понятие *корневой вершины* [12, 18] графа (V, E), обозначим ее O_0, $O_0 \in V$.

Корневая вершина, как и любой объект классификации, является тройкой:

$$O_0 = (A^0, P_0, G_0),$$

где A^0 – множество наименований атрибутов классификатора K.

Пример объекта: $O_{14} : (A^4 = \{4, высшее\}, P_1, G_1 = \{O_{11}, \ldots, O_{1n}\})$.

Каждый *объект классификации* O_i обладает множеством атрибутов A^i, таким что, существует однозначное отображение f множества A^0 на множество A^i:

$$f : A^0 \to A^i, \ \forall j = \overline{1,n}, n = |\, A^0 \,| : a_j^i = f(a_j^0).$$

Под отображением f будем понимать следующее: каждому элементу множества A^0 ставится в соответствие элемент множества A^i с тем же порядковым номером элемента:

$$A^0 = \{код, наименование\}$$

$$A^1 = \{1, мужской\}$$

$$A^2 = \{2, женский\}$$

$$\forall j = \overline{1,n}, n = \mid A^0 \mid = 2:$$

$$a_1^1 = 1 \rightarrow a_1^0 = код, a_2^1 = мужской \rightarrow a_2^0 = наименование$$

$$a_1^2 = 2 \rightarrow a_1^0 = код, a_2^2 = женский \rightarrow a_2^0 = наименование$$

P_0 – множество признаков классификации, которое определяет первый уровень вложенности классификатора K.

G_0 – классификационная группировка нулевого уровня, к которой относятся объекты классификации первого уровня.

Два классификатора K_1 и K_2 *не пересекаются*, если не пересекаются множества их вершин. Таким образом, каждый классификатор K представляет собой множество классификационных объектов, отвечающих различным значениям классификации G.

Как уже говорилось выше, классификаторы могут быть линейными, иерархическими, фасетными. Рассмотрим представление данных типов классификаторов на описанной модели.

Линейный классификатор, или просто справочник, представляет собой множество объектов классификации, которые находятся на первом и единственном уровне вложенности (рис. 1).

Фасетный классификатор представляет собой множество объектов классификации, разбитых на *фасеты*, которые представляют собой первый уровень вложенности; на втором уровне вложенности находятся объекты классификации, принадлежащие тому или иному фасету. Для наглядности переобозначим объекты классификации первого уровня вложенности $O_i = F_i$ (рис. 2). Каждый фасет представляет собой линейный или иерархический классификатор (пример – ОКИН, общероссийский классификатор информации о населении).

Иерархический классификатор представляет собой множество объектов классификации разбитых на классификационные группировки, которые имеют строгую иерархию подчиненности. На первом уровне вложенности находятся вершины иерархии; в классическом варианте вершина должна быть одна, в реальности их может быть больше (рис. 3).

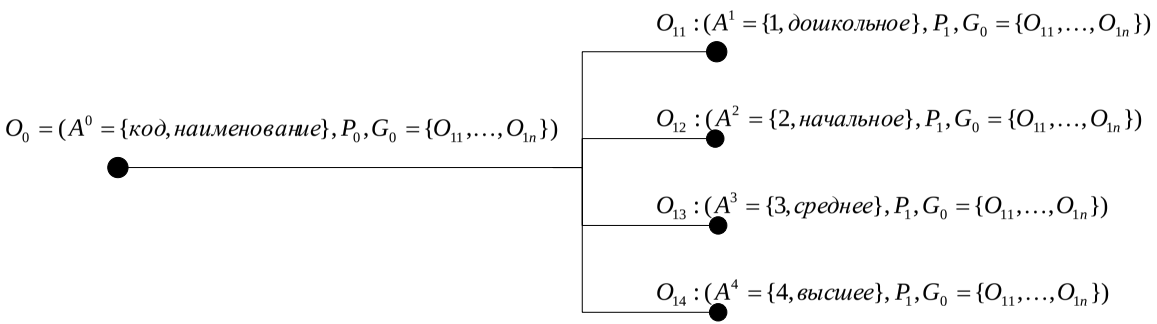


Рис. 1. Представление линейного классификатора

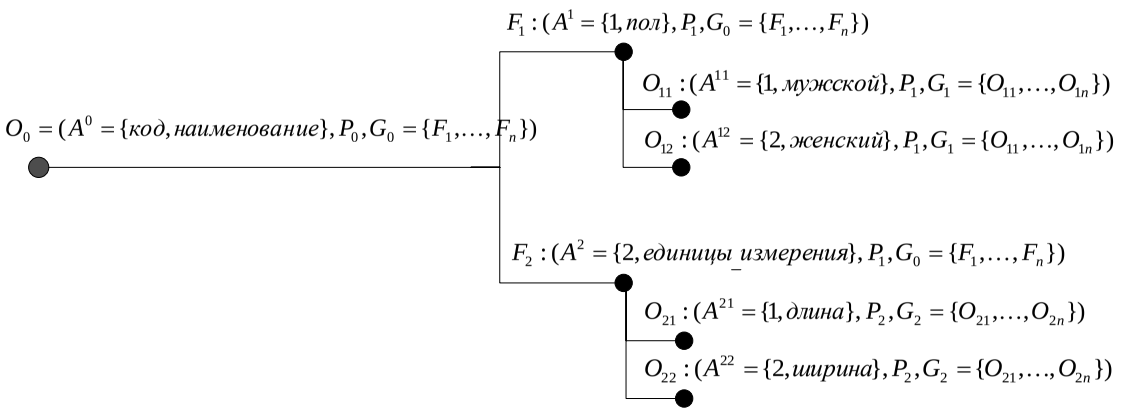


Рис. 2. Представление фасетного классификатора

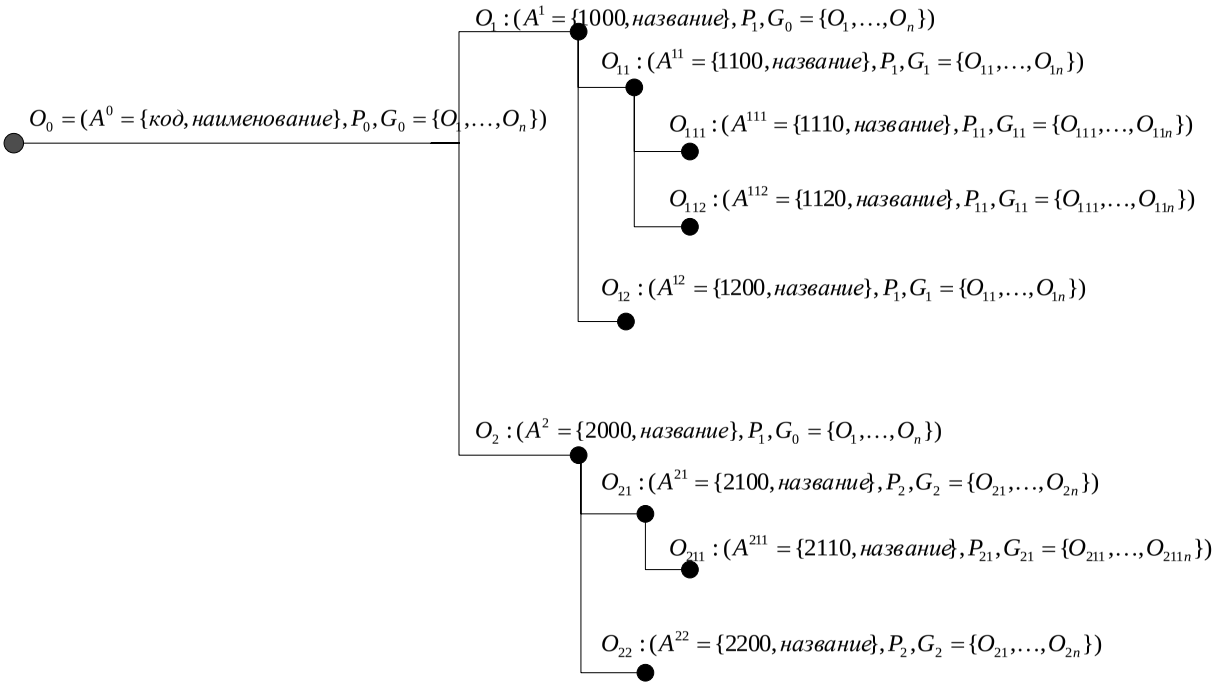


Рис. 3. Представление иерархического классификатора

Для любых двух классификационных объектов $O_1 \in K$ и $O_2 \in K$, $K = (V, E)$, говорят, что O_2 *ссылается на* O_1, или O_2 *подчинен* объекту O_1, и пишут $O_2 E O_1$, если множество признаков классификации X_i объекта O_1 включает в себя множество признаков классификации X_{ij} объекта O_2, т.е. если в графе $K = (V, E)$ есть дуга, соединяющая вершину в O_1 с вершиной O_2. Для каждого классификатора множество отношений соподчинённости представляет множество дуг E в графе $K = (V, E)$.

На рис. 4 представлен фрагмент графа классификатора $K = (V, E)$, где $V = \{O_i, O_{i1}, O_{i2}\}$, $E = \{(O_i, O_{i1}), (O_i, O_{i2})\}$.

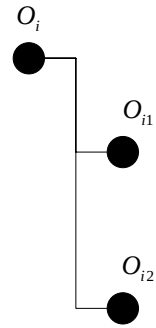


Рис. 4. Фрагмент графа классификатора

Операции над классификационными объектами

Для начала рассмотрим операции, которые можно выполнять над атрибутами классификационного объекта.

– *Добавление атрибута*. Добавление атрибута рассматривается как теоретико-множественная операция объединения двух множеств:

$$Add(O_i, a) : A^{i/} = A^i \cup \{a\}.$$

– *Удаление атрибута*. Удаление атрибута рассматривается как теоретико-множественная операция вычитания двух множеств:

$$Del(O_i, a) : A^{i/} = A^i \setminus \{a\}.$$

– *Обновление атрибута*. Обновление атрибута рассматривается как две операции, удаление старого атрибута и добавление нового:

$$Update(O_i, a, a^{/}) : Del(O_i, a) \,\&\, Add(O_i, a^{/}).$$

Над однотипными классификационными объектами можно выполнять любые теоретико-множественные операции, в том числе:

1. *Объединение*:

$$V_1 \cup V_2 = \{(\{O_{1i}\} \cup \{O_{2j}\}) : (\exists O_{1i} : \{O_{1i}\} \in V_1) \,\&\, (\exists O_{2i} : \{O_{2i}\} \in V_2)\}$$

2. *Вычитание*:

$$V_1 - V_2 = \{(\{O_{1i}\} - \{O_{2j}\}) : (\exists O_{2i} : \{O_{2i}\} \in V_1) \,\&\, (\exists O_{2i} : \{O_{2i}\} \in V_2)\}$$

Введем операцию *манипулирования* классификационными объектами, записываемую в форме $name(O_i, O_i^{/})$, где *name* – имя операции, $O_i, O_i^{/}$ – объекты классификации. Результатом операции является объект классификации. Все объекты классификации, участвующие в операции, однотипные.

3. *Обновление объекта классификации*. Обновление объекта классификации – это замена атрибута O_i на соответствующий атрибут $O_i^{/}$:

$$Update\_O(O_i, O_i^{/}) : (\forall a_i \in A, \exists! a_i^{/} \in A^{/}) \rightarrow (Update(O_i, a_i, a_i^{/})).$$

Над однотипными отношениями соподчиненности можно выполнять любые теоретико-множественные операции, в том числе:

4. *Объединение*:

$$E_1 \cup E_2 = \{(\{(O_{1ik}; O_{1il})\} \cup \{(O_{2jn}; O_{2jm})\}) : (\exists (O_{1ik}; O_{1il}) : (O_{1ik}; O_{1il}) \in \{(O_{1ik}; O_{1il})\})$$
$$\& (\exists (O_{2jn}; O_{2jm}) : (O_{2jn}; O_{2jm}) \in \{(O_{2jn}; O_{2jm})\})\}$$

5. *Вычитание*:

$$E_1 - E_2 = \{(\{(O_{1ik}; O_{1il})\} - \{(O_{2jn}; O_{2jm})\}) : (\exists (O_{2jn}; O_{2jm}) : (O_{2jn}; O_{2jm}) \in \{(O_{1ik}; O_{1il})\})$$
$$\& (\exists (O_{2jn}; O_{2jm}) : (O_{2jn}; O_{2jm}) \in \{(O_{2jn}; O_{2jm})\})\}$$

6. *Добавление отношения соподчиненности*. Добавление отношения соподчиненности рассматривается как теоретико-множественная операция объединения двух множеств:

$$Add(E, (O_i, O_j) : E^{/} = E \cup \{(O_i, O_j)\}.$$

7. *Удаление отношения соподчиненности*. Удаление отношения соподчиненности рассматривается как теоретико-множественная операция вычитания двух множеств:

$$Del(E, (O_i, O_j) : E^{/} = E \setminus \{(O_i, O_j)\}.$$

Над однотипными классификаторами можно выполнять любые теоретико-множественные операции, в том числе:

8. *Объединение*:
$(V_1, E_1) \cup (V_2, E_2) = \{(V, E),\ где\ V = (V_1 \cup V_2)\ \ и\ \ E = (E_1 \cup E_2)\}$.

9. *Вычитание*:
$(V_1, E_1) \setminus (V_2, E_2) = \{(V, E),\ где\ V = (V_1 \setminus V_2)\ \ и\ \ E = (E_1 \setminus E_2)\}$.

Введем операции *манипулирования* классификаторами, записываемые в форме $name(K, O_i)$, где *name* – имя операции, K – классификатор, O_i – объект классификации. Результатом операции является классификатор, а в операции Search результатом является объект классификации. Все объекты классификации, участвующие в операции, однотипные.

10. *Добавление объекта классификации*. Добавление объекта классификации рассматривается как теоретико-множественная операция объединения двух множеств. Операция возвращает сам классификатор, если такой объект классификации в нем уже есть, или объединение уже существующих объектов классификации с добавляемым:

$$Insert(K(V, E), O) = \begin{cases} K, если O \in V \\ V \cup \{O\} and E \cup \{O\} иначе \end{cases}.$$

Добавление объекта в классификатор может быть выполнено поразному. Например, если классификатор является ОК, то для того, чтобы поддерживать интегрируемость, при передаче данных нужно отслеживать, какие объекты классификации являются первоначальными, а какие по мере необходимости были дополнены пользователями. В противном случае может получиться, что один и тот же объект классификации определяет разные информационные объекты.

11. *Удаление объекта классификации*. Удаление объекта классификации рассматривается как теоретико-множественная операция разности двух множеств. В результате выполнения операции возвращается сам классификатор, если такого объекта классификации в нем нет, или из существующих в классификаторе объектов удаляется указанный объект классификации:

$$Delete(K(V, E), O) = \begin{cases} K, если O \notin V \\ V - \{O\} and E - \{O\} иначе \end{cases}.$$

Операция удаления может также выдавать разные результаты. Например, если классификатор является ОК, то удалять можно только те объекты классификации, которые ввёл пользователь. Иначе может возникнуть противоречивость классификаторов, находящихся на разных рабочих местах.

12. *Обновление объекта классификации*. Операция *Update* обновляет объект классификации из классификатора, а именно: обновляет значения атрибутов объекта классификации. Операция возвращает Λ – *неопределенное значение*, если объект классификации обновлять не надо, или обновляет значения атрибутов объекта классификации, используя операцию *Update_O*.

$$Update(K(V,E),O_i,O_j)=\begin{cases}\Lambda, & если \quad O_i(A)\notin O_j(A)\\ & иначе \quad update\_O(O_i,O_j)\end{cases}\}.$$

Операция обновления тоже имеет свои особенности, так как обновление классификатора может идти полностью, т.е. одна версия классификатора заменяется другой, или же может быть выполнено частичное обновление каких-то конкретных объектов классификации.

13. *Операция поиска*:

$$Search(K(V,E),O)=\begin{cases}O, если O\in V\\ \Lambda\_иначе\end{cases}.$$

Операция *Search* позволяет найти объект классификации в классификаторе. Операция возвращает сам объект классификации, если он есть в классификаторе, или, если объекта классификации нет, то Λ – *неопределённое значение*.

Операции над классификаторами

Рассмотрим основные алгоритмы работы с классификаторами. Можно выделить *четыре основных операции по манипулированию классификаторами* в информационных системах: *создание*, *удаление*, *обновление*, *импорт*/*экспорт* классификаторов. Опишем алгоритмы выполнения этих операций на основе предложенной модели.

1. *Создание классификатора*. Под созданием классификатора будем понимать последовательное создание множества объектов классификации, в том числе признаков классификации, классификационных группировок, и отношений соподчиненности между объектами классификации. В данной интерпретации не рассматриваются задачи

выявления закономерностей на основе массивов данных, классификации исходных данных (разработка кодов классификации). Будем считать, что этот этап уже выполнен экспертами или автоматизированными системами, то есть в нашем случае создание классификатора сводится к двум этапам: созданию множества объектов классификации и созданию множества отношений соподчинённости.

Создание множества объектов классификации может быть выполнено

- экспертом вручную;
- импортом уже созданных объектов классификации из внешних источников.

Создание множества отношений соподчинённости может быть выполнено

- экспертом вручную;
- импортом уже созданных отношений соподчинённости из внешних источников;
- на основе существующих в модели признаков классификации и классификационных группировок.

2. *Удаление классификатора*. Под удалением классификатора будем понимать последовательное удаление множества отношений соподчиненности между объектами классификации и множества объектов классификации. Данная операция не является тривиальной, поэтому на ней акцентироваться внимание не будет.

3. *Обновление классификатора*. Под обновлением классификатора будем понимать последовательное изменение множества объектов классификации, в том числе признаков классификации, классификационных группировок, и множества отношений соподчиненности между объектами классификации. Как уже говорилось выше, будем рассматривать три операции:

- Добавление во множество объектов классификации нового элемента.
- Изменение атрибутов у элемента множества объектов классификации:
  - изменение значений атрибутов;
  - удаление атрибутов;
  - добавление атрибутов.
- Удаление элемента из множества объектов классификации.

Введем обозначения: K_{update} – изменения в классификаторе, которые нужно произвести; K_{del} – подмножество объектов классификации и отношений соподчиненности, которые нужно удалить; K_{add} –

подмножество объектов классификации и отношений соподчиненности, которые нужно добавить в классификатор; K_{upd} – подмножество объектов классификации, которые нужно обновить; K_{out} – классификатор из внешних источников; K – классификатор соответствующий универсальной модели; $a_j \sim a_i$ – элемент a_j эквивалентен элементу a_i; $a_j \to a_i$ – элемент a_j ставится в соответствие элементу a_i; a_i^* – элемент является обязательным элементом множества A^0.

Следовательно, обновление классификатора можно разделить на следующие этапы:

1) удалить подмножество K_{del} из классификатора K;
2) если необходимо изменить множество атрибутов A^0 классификатора K;
3) добавить подмножество K_{add} в классификатор K;
4) обновить объекты классификации из подмножества K_{upd}.

4. *Импорт классификатора*. Под импортом классификатора будем понимать создание множества объектов классификации и множества отношений соподчиненности на основе сторонней модели. Под сторонней моделью будем понимать модель, описывающую данные из внешних источников. Перед созданием множеств возможно приведение (отображение) множества атрибутов сторонней модели на множество атрибутов предложенной модели.

5. *Экспорт классификатора*. Под экспортом классификатора будем понимать создание отдельных множеств объектов классификации и отношений соподчиненности в соответствии со сторонней моделью. Перед созданием множеств возможно приведение (отображение) множества атрибутов предложенной модели на множество атрибутов сторонней модели. Экспортироваться может не весь классификатор, а только его часть, следовательно, при экспорте необходимо учитывать условия экспорта данных.

Заключение

Разработанный на основе описанной модели комплекс программ включает средства «ручного» ведения классификаторов с возможностью задания их структуры, описаний элементов, их ввода и редактирования. Кроме того, реализованы средства импорта классификаторов, представленных в формате таблиц MS Excel или реляционных баз данных – источников данных, доступных через ODBC.

Для конечного пользователя реализованы удобные средства навигации, визуализации НСИ, просмотра и поиска информации.

Разработанная модель может также стать основой для реализации средств автоматизации разработки онтологий предметных областей на основе НСИ, классификаторов (международных, общероссийских, отраслевых).

Комплекс программ ведения НСИ [4, 5, 15, 16, 17] позволяет настраиваться на различные программные платформы, работать под управлением различных операционных систем Microsoft, использовать для работы различные реляционные СУБД и источники данных, для которых существуют драйверы ODBC.

Для функционирования runtime-компонентов необходимо установить .NET Framework (распространяется бесплатно), драйверы ODBC (при установке операционной системы) и СУБД (можно использовать, в частности, Microsoft SQL Server Express, которая распространяется бесплатно). Данные НСИ могут быть импортированы, в частности, из электронных таблиц Excel.

Ограничения использования системы определяются только требованиями лицензионной чистоты и требованиями, предъявляемыми перечисленными выше программными средствами.

Библиографический список

1. *Вендров А.М.* Проектирование программного обеспечения экономических информационных систем. М.: Финансы и статистика, 2000.
2. ГОСТ 50.1.021-2000 Единая система классификации и кодирования технико-экономической информации. М.: Изд. стандартов, 2000.
3. ГОСТ 34.601-90. Автоматизированные Системы Стадии создания. Комплекс стандартов на автоматизированные системы. М.: Изд. стандартов, 1997.
4. *Лукиных Н.В.* Реализация унифицированных средств для работы с классификаторами в информационных системах // Технологии Microsoft в информатике и программировании: тезисы докладов конференции-конкурса работ студентов, аспирантов и молодых ученых. Новосибирск, 2005. С. 39-40.
5. *Лукиных Н.В.* Универсальный компонент управления для представления и администрирования классификаторов // Технологии Microsoft в информатике и программировании:

тезисы докладов конференции-конкурса работ студентов, аспирантов и молодых ученых. Новосибирск, 2006. С. 109-111.

6. *Лядова Л.Н., Скрябина Н.В*. Математическая модель для предоставления классификаторов // Математика программных систем: межвуз. сб. науч. тр. / Перм. ун-т. Пермь, 2003. С. 56-60.
7. *Лядова Л.Н*. Архитектура информационной системы «Образование Пермской области» // Математика программных систем: межвуз. сб. науч. тр. / Перм. ун-т. Пермь, 2002. С. 25-35.
8. *Лядова Л.Н., Рыжков С.А*. CASE-технология METAS // Математика программных систем: межвуз. сб. науч. тр. / Перм. ун-т. Пермь, 2003. С. 4-18.
9. *Лядова Л.Н., Рыжков С.А*. METAS – технология создания информационных систем сферы образования // XIV Международная конференция-выставка «Информационные технологии в образовании»: сборник трудов участников конференции. Ч. IV. М.: МИФИ, 2004. С. 38-40.
10. *Лядова Л.Н., Рыжков С.А*. Технология создания информационных систем, управляемых метаданными // Математические методы и информационные технологии в экономике, социологии и образовании: сборник статей XIII Международной научно-технической конференции / Пенза: Пензенская государственная технологическая академия, Приволжский дом знаний, 2004. С. 316-319.
11. *Рыжков С.А*. Концепция метаданных в разработке информационных систем // Математика программных систем: Межвуз. сб. науч. тр. / Перм. ун-т. Пермь, 2002. С. 36-44.
12. *Оре О*. Теория графов. М.: Наука, 1980.
13. *Помазков Я*. Системы НСИ: мировой опыт и тенденции развития // Компьютерный журнал PCWeek/RE. №12/2006. С. 57-61.
14. *Свами М., Тхуласироман К*. Графы, сети и алгоритмы. М.: Мир, 1984.
15. *Скрябина Н.В*. Автоматизация построения классификаторов на основе анализа документов // Материалы XLIII Международной научной студенческой конференции «Студент и научно-технический прогресс»: Информационные технологии / Новосибирск: Новосиб. гос. ун-т, 2005. С. 210-211.

16. *Скрябина Н.В*. Математическая модель для предоставления классификаторов // Современные проблемы механики и прикладной математики: сборник тр. междунар. школы-семинара / Воронеж: ВГУ, 2004. С. 230-235.
17. *Скрябина Н.В*. Унифицированные средства управления классификаторами в информационных системах // Молодежь. Образование. Экономика: сборник научных статей участников конференции. Часть 4. – Ярославль: Изд-во «Ремдер», 2004. С. 170-174.
18. *Яблонский С.В*. Введение в дискретную математику. 2-е изд. М.: Наука, 1986.

А.И. Миков, Н.З. Нгуен

Кубанский государственный университет

alexander_mikov@mail.ru, duycoi90@gmail.com

ИССЛЕДОВАНИЕ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ ПРИ ИМИТАЦИОННОМ МОДЕЛИРОВАНИИ СЕТЕЙ БОЛЬШОГО МАСШТАБА

Введение

Процесс имитационного моделирования дискретных систем является довольно затратным по времени. Для получения достаточно статистически устойчивых оценок параметров исследуемой системы требуется провести много имитационных экспериментов. При сложной архитектуре системы, например сети массового обслуживания большого размера, обрабатываются десятки и сотни миллионов событий, поэтому, очень важной является *оптимизация построения симулятора* – основной системной программы, обеспечивающей управление очередями событий. Ускорению работы может способствовать распараллеливание процессов при наличии нескольких устройств обработки.

В данной работе рассматриваются *событийно-ориентированные системы моделирования*, в которых

- продвижение модельного времени происходит от события к событию;
- произошедшие события определяют последовательность смены состояний, связанных с наступлением этих событий.

При анализе работы систем имитационного моделирования различают [1, 2]: *физическое* время, используемое в моделируемой физической системе, *модельное* время (представление физического времени в модели) и *процессорное* время (время работы симулятора).

Во время работы имитационной модели формируется *список событий*, упорядоченных *по возрастанию времени* их наступления. Кро-

ме моментов времени их наступления события характеризуются *причинно-следственными связями* между ними, т.е. существуют события, зависящие друг от друга, и, как правило, они не могут обрабатываться одновременно (параллельно), а только в порядке, определяемом соответствующими причинно-следственными связями. Таким образом, в любой момент процессорного времени в системе имитационного моделирования имеется *множество запланированных событий*, на котором определены два отношения: линейное *отношение предшествования по времени* и частичное *отношение причинно-следственных связей*.

Если система имитационного моделирования исполняется на компьютере с n процессорами, то это множество должно быть разделено, по возможности, на n равных частей для достижения максимального ускорения процесса. Основная сложность заключается в том, что множество динамично, оно изменяется с каждым новым обработанным событием, поэтому алгоритм распределения событий на обработку по процессорам должен быть очень экономичным.

В работе приводятся три различных алгоритма распределения, и производится анализ их эффективности на примерах сетей массового обслуживания (СМО) определённой конфигурации.

Алгоритм и модель

Принцип алгоритма распределения событий для обработки по n процессорам заключается в следующем. В начальный момент модельного времени в системе одновременно появляется определенное количество событий, запись о каждом из них имеет следующие поля: *номер* (идентификатор) *события*; *время наступления события*; *порядковый номер заявки*, к которому это событие относится; *номер узла*, на котором эта заявка обрабатывается. Эти записи образуют *список запланированных событий*, упорядоченных по моменту их появления. Так как нужно обработать эти события по порядку, то очевидно, что должен быть определен *таймер модельного времени* для фиксации текущего модельного времени. Следует отметить, что условием, вызывающим приращение таймера, является наступление времени «ближайшего события». Ближайшее событие – это то событие, возникновение которого запланировано на момент времени, равный минимальному значению модельного времени в списке запланированных событий. Это делается для того, чтобы избежать обработки промежуточных моментов времени, на которые не планируется выполнение никаких событий.

После получения списка событий в некоторый определенный момент времени t необходима их обработка n процессорами. Также нужно отметить, что обработка событий происходит в процессорном вре-

мени, которое не связано однозначно с модельным временем. Приращение таймера процессорного времени происходит тогда, когда все события в момент *t* ещё не будут обработаны, а также, когда все процессоры в этот момент заняты. Кроме этого нужно учесть, что в один и тот же момент модельного времени могут появиться события, обладающие причинно-следственной связью. Следовательно, приращение можно также получить, если оставшиеся в списке события зависят от тех, которые в данный момент находятся в обработке. Моделирование завершится в тот момент, когда количество обработанных заявок превысит некоторое число, заданное пользователем.

Имея *n* процессоров для обработки событий, можно определить следующие методы их распределения в системе:

1. Все *n* процессоров – общие для всех узлов модели.
2. Модель делится на *k* частей (в зависимости от количества узлов *m*). Имеющиеся процессоры распределяется поровну между этими частями, т.е. на каждый узел приходится *n/m* процессоров.
3. Узлы имеют одинаковое количество процессоров на обработку, т.е. имея *m* узлов и *n* процессоров, получаем, что каждый узел имеет по *n/m* процессоров для обработки.

Используя эти методы, сравним результаты работы соответствующих им алгоритмов для различных моделей.

Определим класс моделей для последующего анализа.

Рассмотрим класс сетей СМО с ожиданием (неограниченной очередью), размерностью *k*×*k* следующего вида (рис. 1), где круг – это узел или сервер, прямоугольник – очередь с неограниченным количеством мест для ожидания.

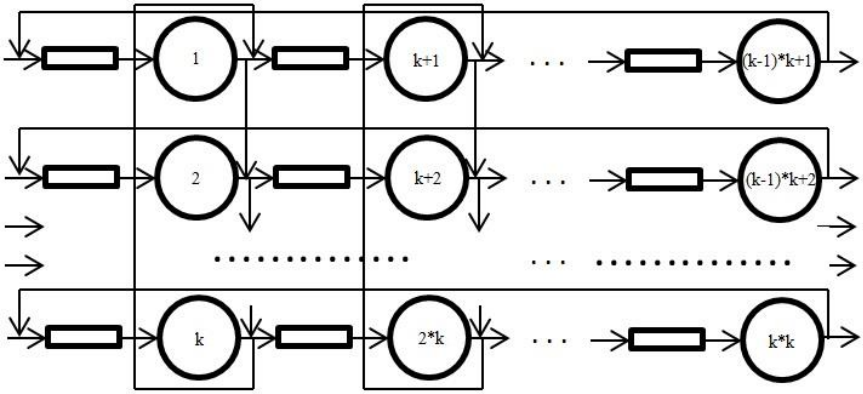


Рис. 1. Структура СМО размерности *k*×*k* с ожиданием

Следует отметить, что каждый узел имеет три вида событий, связанныз с обслуживанией на нём заявки, при наступлении которых в какой-либо момент времени система изменяет свое состояние: приход заявки; начало обработки заявки; конец обработки заявки.

Каждый *i*-й узел данной сети имеет два пути передачи заявки на последующую ее обработку, подчиняясь следующему закону:

$$W_{1,2}(i) = \begin{cases} (k+i;\, i+1-k), \text{если } i \ mod \ k = 0 \\ (i-(k-1)\times k;\, 0), \text{если } i > (k-1)\times k \\ (k+i;\, k+i+1), \text{в противном случае} \end{cases}.$$

Здесь $W_{1,2}(i)$ – функция перехода, значением которой является пара чисел, эквивалентных соответствующим узлам системы, где ‘0’ означает выход из системы. Следует отметить, что передача заявки происходит по одному из двух путей с равной вероятностью.

Проведем анализ поведения данного класса систем размерности $k \times k$ ($k = \overline{2,6}$) при использовании трех методов и при обработке 1 млн. заявок. На рис. 2 приведены графики зависимости времени *t* обработки заявок от количества процессоров *n*. Время указано в миллионах условных единиц.

Исследуя данные графики можно утверждать, что метод 1 работает лучше, чем два остальных алгоритма, и стремится к минимальному значению времени обработки быстрее, нежели другие. Для структур с $k = 2$, $k = 4$, $k = 6$ видно, что при увеличении числа процессоров в 2 раза, имея изначально 1 процессор, используя метод 1, скорость выполнения программы можно увеличить приблизительно в 2 раза, тогда как метод 2 даёт уменьшение времени только в 1.5 раза. Также несложно заметить, что минимальное время выполнения можно получить при наличии в системе количества процессоров, равного удвоенному количеству узлов (серверов) в структуре.

Следует отметить, что важной частью исследования является также анализ загрузки и дисбаланса процессоров в течение всего времени моделирования (рис. 3).

Здесь видно, что при использовании метода 1 и метода 2 проявляется большой дисбаланс нагрузки, тогда как метод 3 даёт довольно незначительный дисбаланс. Из этого следует, что метод 3 лучше использует ресурсы имеющихся процессоров. Но значительный минус этого метода в том, что он не может работать при количестве процессоров меньшем количества серверов в структуре сети, так как он может работать только в том случае, когда в каждом сервере находится как минимум 1 процессор для обработки событий.

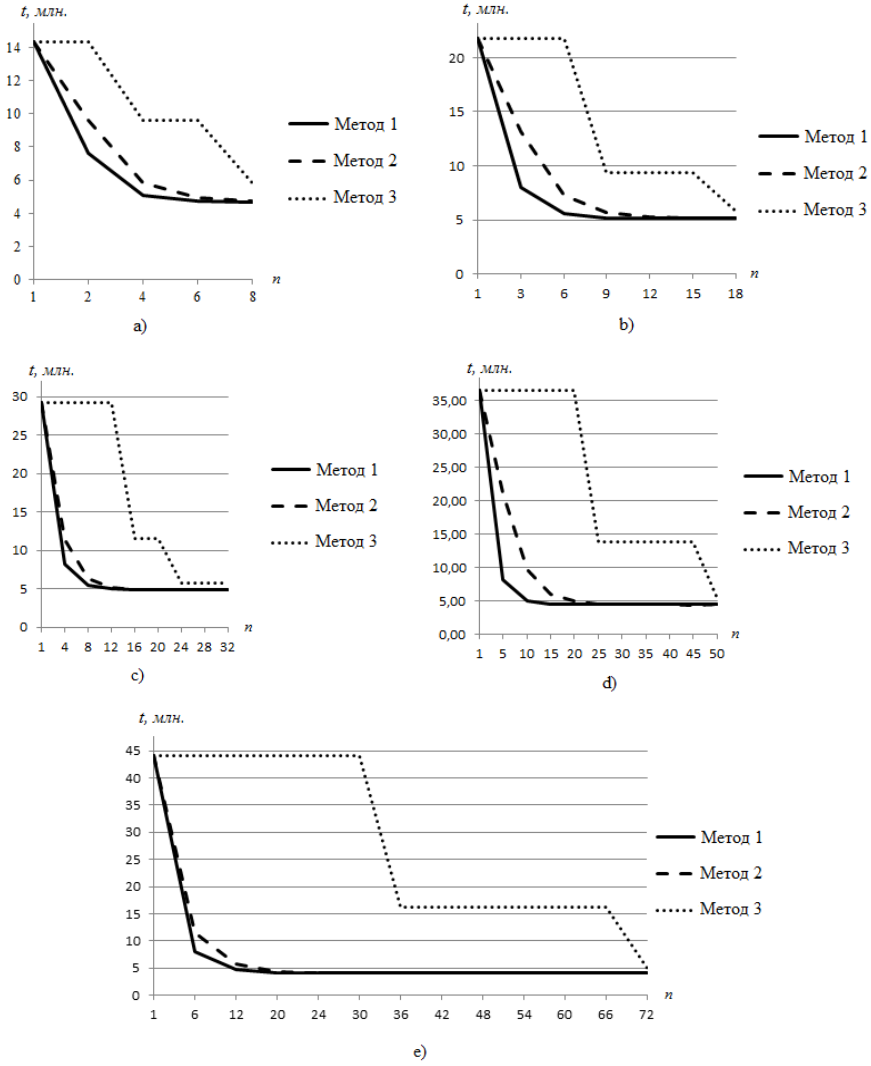


Рис. 2. Зависимости времени обработки от количества процессоров для систем размерности a) 2×2, b) 3×3, c) 4×4, d) 5×5, e) 6×6

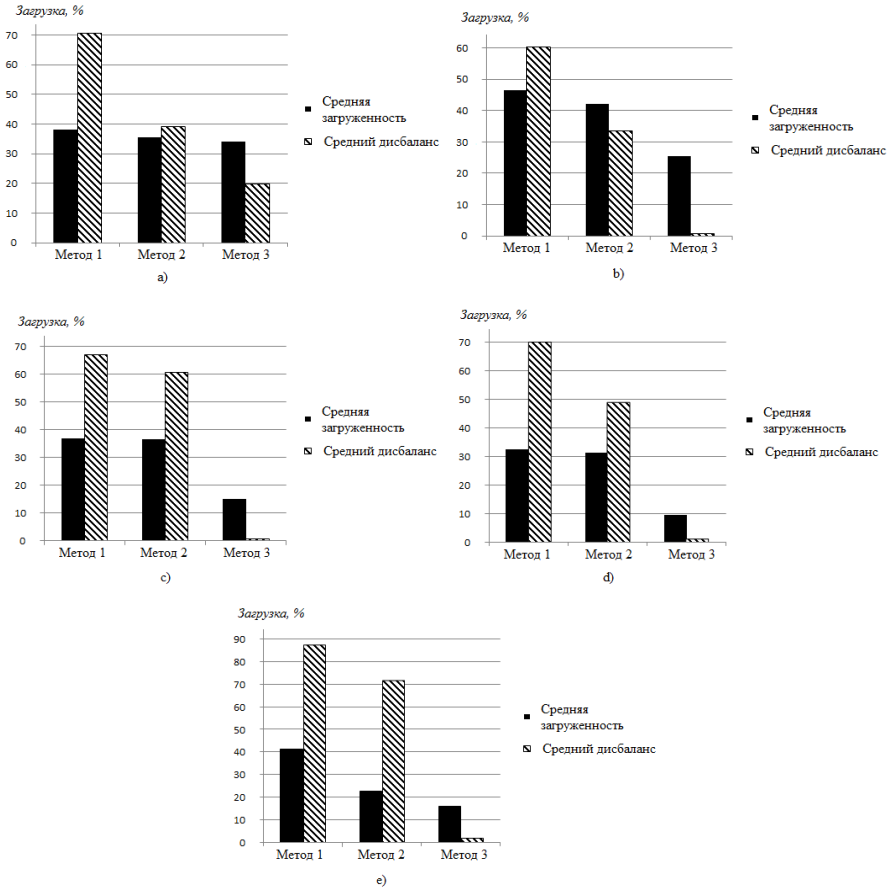


Рис. 3. Диаграммы средней загрузки и дисбаланса процессоров при моделировании систем различной размерности:
a) 2×2, b) 3×3, c) 4×4, d) 5×5, e) 6×6

Анализируя данные, приведённые в табл. 1, заметим, что обслуживание одного миллиона заявок вычислительная структура с параметром $k = 6$ выполнит быстрее, чем другие. При этом нужно учесть, что она обрабатывает событий больше, нежели другие структуры, меньшие по масштабу. Кроме того, нельзя оставить без внимания тот факт, что количество ресурсов, затрачиваемых на обработку, соответственно выше.

Таблица 1. Объем работы для систем разной размерности

Параметр структуры, *k*	Количество событий, обработанных в системе, млн.	Минимальное время моделирования, млн. ед.
2	14,34	4,648
3	21,74	5,115
4	29,18	4,889
5	36,58	4,452
6	44,03	4,051

Заметим, что при обработке 1 млн. заявок получаем уменьшение времени по отношению к последовательной обработке: при $k = 2$ в 3 раза; при $k = 3$ в 4,25 раза; при $k = 4$ в 5,96 раза; при $k = 5$ в 8,21 раза; при $k = 6$ в 10,86 раза.

Заключение

В данной работе представлен сравнительный анализ трех методов обработки списка событий, планируемых во время работы имитационной модели СМО. Было обнаружено, что при малом количестве процессоров (меньшем хотя бы удвоенного количества серверов в модели) лучшие результаты даёт использование метода 1, т.к. он позволяет получить минимальное время обработки с наименьшими затратами ресурсов. В то же время для получения лучшего баланса загрузки процессоров с минимальными затратами метод 2 подходит лучше. При наличии большого количества процессоров все три метода достигают минимального предела времени работы модели, и для обеспечения наилучшего баланса загрузки процессоров следует использовать метод 3. Просчитав результаты работы системы, начиная со структуры 3×3, можно сделать вывод о том, что при увеличении масштаба сети мы получаем большую производительность.

Библиографический список

1. *Миков А.И., Замятина Е.Б.* Инструментальные средства имитационного моделирования для анализа бизнес-процессов и управления рисками // Информатизация и связь. 2011. № 3. С. 14–16.
2. *Замятина Е.Б., Миков А.И.* Программные средства системы имитации Triad.NET для обеспечения ее адаптируемости и открытости // Информатизация и связь. 2012. №5. С. 130–133.

А.О. Сухов

Национальный исследовательский университет
«Высшая школа экономики»
(Пермский филиал)

ASuhov@hse.ru

СРАВНЕНИЕ ЯЗЫКОВ И ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ ТРАНСФОРМАЦИИ ВИЗУАЛЬНЫХ МОДЕЛЕЙ[2]

Введение

Трансформация моделей – центральная часть модельно-ориентированного подхода к разработке программного обеспечения, поскольку существование в системе одной модели, выполненной с разных точек зрения и, возможно, описанной в нотации различных языков моделирования, требует наличия средств преобразования моделей как между различными уровнями иерархии моделей, так и внутри одного уровня: при переходе от одного языка моделирования к другому для построения единой модели системы, описывающей всю систему в целом.

Использование предметно-ориентированных языков (DSL) и инструментальных средств их создания также затрагивает проблему трансформации, поскольку появляется потребность экспорта созданных пользователем моделей во внешние системы, которые, как правило, используют один из стандартных языков моделирования, отличающийся от разработанного DSL. Именно поэтому система MetaLanguage, предназначенная для создания визуальных динамически настраиваемых предметно-ориентированных языков моделирования [30], должна содержать средства описания и выполнения трансформаций визуальных моделей.

[2] Работа выполнена при поддержке РФФИ (проект № 12-07-00763-а)

Требования, предъявляемые к компоненту трансформации системы MetaLanguage, определяются назначением данного инструментария и могут быть сформулированы следующим образом:

- описание трансформаций должно производиться с использованием визуальных языков, понятных различным категориям специалистов;
- возможность описания как горизонтальных, так и вертикальных трансформаций;
- возможность описания горизонтальных трансформаций вида «модель-модель» и «модель-текст»;
- возможность использования метаязыка, разработанного пользователем, для спецификации исходной и целевой метамодели (модели языка моделирования);
- возможность описания преобразований атрибутов и ограничений элементов метамоделей.

Существуют различные подходы к трансформации моделей, некоторые из них имеют формальную основу. Так, системы AGG, GReAT, VIATRA используют для выполнения трансформаций правила переписывания графов (graph rewriting), а другие – применяют технологии из других областей программной инженерии, например подход MTBE основывается на методе программирования по образцу.

Трансформация моделей с использованием графовых грамматик

Графовые грамматики достаточно часто применяются для описания каких-либо преобразований, выполняемых над графами: трансформация визуальных моделей [5], задание операционной семантики моделей [3], анализ программных систем с динамически развивающимися структурами [2] и др. Такие грамматики позволяют наглядно описать преобразования, которые должны происходить в системе при выполнении над ней заданных в грамматике операций.

Базовым понятием при описании трансформации графов является *продукционное правило*, которое имеет вид $p : L \to R$, где p – имя правила, L – *левая часть правила*, которая называется *паттерном*, а R – *правая часть правила*, которая называется *графом замены*.

Графовая грамматика – это пара $GG = (P, G_0)$, где P – набор продукционных правил, G_0 – исходный граф грамматики.

Правила применяются к исходному графу, называемому *хост-графом*. Если граф из левой части правила был найден в исходном графе, то правило может быть применено.

Пусть дан исходный граф G, а также графы L и R, которые представляют собой левую и правую части продукционного правила $p: L \to R$, причем граф L является подграфом графа G. Тогда *применением правила p* к исходному графу G называется замена в графе G подграфа L на граф R, результатом замены является граф H, причём граф R – подграф графа H [18].

Графовая трансформация – это последовательное применение к исходному помеченному графу G_0 конечного набора правил

$$P = (p_1, p_2 \dots p_n): G_0 \overset{p_1}{\Rightarrow} G_1 \overset{p_2}{\Rightarrow} \dots \overset{p_n}{\Rightarrow} G_n,$$

где G_n – результат выполнения трансформации.

По направлению преобразования трансформации можно классифицировать как вертикальные и горизонтальные. *Вертикальные* трансформации управляют ходом преобразования моделей при переходе от одного уровня иерархии к другому, например при отображении объектов метамодели на объекты модели предметной области (рис. 1). *Горизонтальная трансформация* – это преобразование, при котором исходная и целевая модель принадлежат одному уровню иерархии. Примером горизонтальной трансформации является преобразование описания модели, выполненного на одном языке моделирования, в эквивалентное описание на другом языке.

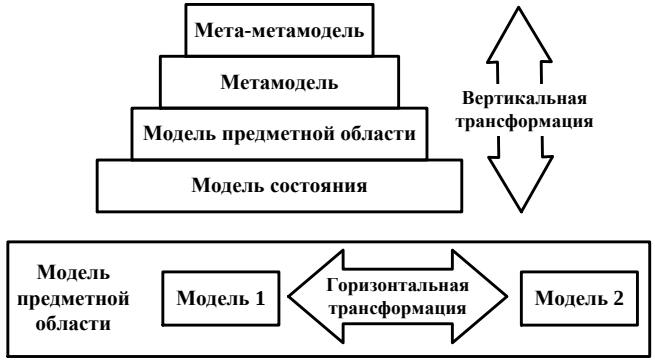


Рис. 1. Вертикальная и горизонтальная трансформации моделей

Для того чтобы выполнить трансформацию моделей необходимо, чтобы они были описаны с использованием некоторых метамоделей. В зависимости от языка, на котором описаны исходная и целевая модель, горизонтальные трансформации можно разделить на два вида:

эндогенные и экзогенные. *Эндогенная трансформация* – это преобразование, при котором исходная и целевая модели описаны на одном языке моделирования. *Экзогенная трансформация* – это преобразование, при котором исходная и целевая модели описаны на различных языках моделирования [25].

Рассмотрим основанные на графовых грамматиках методы трансформации визуальных моделей и реализующие их инструментальные средства.

Attributed Graph Grammar

AGG (Attributed Graph Grammar) – инструментальная среда для описания и выполнения трансформаций типизированных атрибутных графов, реализованная на платформе Java.

Среда AGG состоит из графического интерфейса пользователя, включающего несколько визуальных редакторов, интерпретатора и ряда инструментов проверки допустимости. Визуальные редакторы поддерживают возможность редактирования графов, правил и графовых грамматик.

Благодаря наличию формальной основы – алгебраического подхода к трансформации графов [15], AGG поддерживает возможность проверки допустимости в форме парсинга графа, проверки непротиворечивости графов и правил трансформации.

Граф в AGG представляет собой пару $G = (V, E)$, где V – непустое множество *вершин* графа, E – множество *ребер* графа. Элементы множества, полученного в результате объединения непересекающихся множеств вершин и ребер, называют *объектами* графа.

Следует отметить, что данный инструментарий позволяет проводить кратные ребра, поэтому каждое ребро, как и вершина, имеет уникальный идентификатор.

Каждый элемент графа в AGG имеет определенный тип из заданного множества типов, которое состоит из двух подмножеств: набора типов вершин и набора типов ребер. Создание элементов, не имеющих определенного типа, в AGG запрещено. Тип характеризуется его именем, которое может быть пустым, и графическим представлением, которое определяет форму, цвет, метку вершины или ребра, причем если имена двух типов совпадают, но их графические представления отличны, то соответствующие типы различны.

Описание атрибута в AGG аналогично объявлению переменной в стандартном языке программирования: пользователь должен задать имя и тип атрибута, после чего может присваивать ему любые значения указанного типа. Все элементы атрибутных графов AGG должны иметь ровно одно значение для каждого атрибута. При описании атри-

бутов пользователь может использовать не только простые типы, такие как символы, строки или целые числа, но и любой произвольный класс, описанный на языке Java.

После описания графов моделей пользователю необходимо задать правила их преобразования. Среда AGG поддерживает два способа определения трансформаций: подход одинарного выталкивания (SPO) [16], в этом случае правило состоит из левой (LHS) и правой (RHS) части вместе с частичным морфизмом графа $r : LHS \to RHS$, и подход двойного выталкивания (DPO) [12], который можно рассматривать как полный морфизм графа с дополнительным графом склеивания I, т.е. $r : LHS \leftarrow I \to RHS$, причем частичные морфизмы графа не обязательно должны быть инъективны.

Правило может содержать отрицательные условия применения правила (NAC), определяющие контекст, в котором правило не должно срабатывать.

LHS правила и NAC в качестве значений атрибутов могут содержать лишь константы и переменные, но не Java-выражения. Это ограничение позволяет устанавливать соответствие атрибутов простым сравнением значений констант или переменных, однако атрибуты RHS правила могут содержать выражения, описанные на языке Java. При описании NAC пользователь может использовать переменные, объявленные в LHS, или переменные, объявленные как входные параметры. Область видимости переменной – это правило, в котором она описана.

В общем случае может быть найдено несколько соответствий LHS в хост-графе либо может не существовать ни одного соответствия. В первом случае должно быть выбрано только одно из найденных соответствий, этот выбор зависит от контекста применения правила, а также от настроек системы: будет ли сделан выбор автоматически или при участии пользователя. В случае, если не было найдено ни одного соответствия, правило не может быть применено к данному графу.

Правила соответствий могут быть не инъективными, тогда несколько объектов левой части правила будут отображены на один единственный объект хост-графа. Однако такие не инъективные соответствия способны вызвать конфликты при удалении и сохранении объектов графа. Эти конфликты могут быть разрешены заданием приоритета операции удаления над операцией сохранения в подходе SPO или неприменимостью правила в подходе DPO, поскольку условие склеивания не будет выполнено.

В результате применения правила $r : LHS \to RHS$ паттерн, найденный для LHS, будет заменен RHS-частью правила. Соответствие

$m : LHS \to G$ является полным морфизмом, так как для любого объекта o в LHS найдется его образ $m(o)$ в хост-графе G. Если для объекта o существует образ $r(o)$ в RHS, то его образ $m(o)$ в хост-графе будет сохранен во время преобразования, иначе будет удален. Объекты RHS, не имеющие прообразов в LHS, создаются во время преобразования. Объекты хост-графа, которые не содержатся в соответствии, вообще не участвуют в применении правила, они лишь формируют контекст, который сохраняется во время преобразования. Если условие склеивания установить не удалось, то AGG для трансформации использует подход SPO; в этом случае висячие ссылки будут удалены в результате выполнения преобразования, даже в случае их принадлежности контексту. С другой стороны, если условие склеивания установлено, AGG применяет подход DPO.

Помимо манипулирования вершинами и ребрами графа, правило трансформации может также выполнять вычисление атрибутов, которое не ограничивается применением простых арифметических операций – пользователь может вызывать любые методы, описанные на языке Java.

AGG позволяет пользователю разделять правила по уровням и тем самым определять порядок применения правил. Интерпретатор сначала должен выполнить все правила нулевого уровня, пока это возможно, далее – все правила первого уровня и т.д. Если все правила самого последнего уровня были выполнены, происходит остановка процесса преобразования. Благодаря разделению правил на уровни, появляется возможность задания простого потока управления преобразованием графа.

AGG предоставляет в распоряжение пользователя парсер графа, который способен проверить принадлежность графа языку, определяемому графовой грамматикой. Данная задача эффективно разрешима лишь для ограниченных классов графовых грамматик. Для того чтобы проверить, принадлежит ли граф некоторому языку, необходимо инвертировать все правила грамматики. В результате применения полученных правил в определенном порядке все графы некоторого языка должны быть редуцированы к стартовому графу грамматики.

Однако AGG не может автоматически инвертировать все правила грамматики, поэтому синтаксический анализатор ожидает вместо порождающей грамматики грамматику парсинга, содержащую редуцирующие правила и граф остановки. AGG пытается применить правила парсинга так, чтобы хост-граф был редуцирован к графу остановки. Если это удается сделать, то хост-граф принадлежит языку, определенному грамматикой, и существует последовательность преобразований

хост-графа, приводящая к графу остановки, иначе граф не принадлежит грамматике.

AGG реализует алгоритм парсинга, основанный на переборе с возвратом, при этом парсер создает дерево вывода возможных редукций хост-графа с тупиковыми графами, к которым больше нельзя применить ни одного правила.

Преобразование графа может быть выполнено в двух различных режимах: режиме отладки и режиме интерпретации. Режим отладки позволяет применить выбранное правило только к текущему хост-графу. Соответствующий морфизм может быть определен пользователем. Поскольку задание соответствия «вручную» – достаточно сложная работа, AGG поддерживает возможность автоматического завершения частичных соответствий. Если существует несколько вариантов завершения, то все они будут вычислены и показаны один за другим в редакторе графа. После выбора пользователем соответствия правило будет применено к хост-графу. Режим интерпретации является более сложным и применяется не только к одному правилу, но и к целой последовательности правил. Правило применяются столько раз, сколько это возможно, пока соответствие для другого правила не будет найдено. Следует отметить, что результирующий граф не является единственным возможным, поскольку применение одного правила может повлечь за собой отказ от выполнения другого.

AGG поддерживает возможность обмена графами в формате GXL, разработанном на основе XML. Хост-граф может быть сохранен в GXL, а грамматика графа – в формате XML, который в дальнейшем авторы планируют заменить специальным форматом GTXL [23]. Благодаря такому подходу появляется возможность выполнять трансформации моделей, описанных с помощью некоторого CASE-средства, предварительно сохранив их в формате GXL.

Подводя итог, можно гвоорить о том, что AGG – инструментальное средство описания графовых грамматик, которое поддерживает трансформацию графов. AGG использует для описания исходной и целевой моделей ориентированные типизированные атрибутные графы, что позволяет применять данный инструментарий практически в любой предметной области. Использование в качестве формальной основы алгебраического подхода к трансформации графов позволяет произвести парсинг графа, проверить графовую модель на непротиворечивость.

Расширение возможностей графовых грамматик средствами языка Java позволяет приблизить формальную модель к предметной области и реализовать сложные функции поведения системы.

Преобразование модели задается правилами переписывания графа, которые применяются недетерминировано до тех пор, пока ни одно из них не может быть больше применено. Если необходимо явно указать порядок применения правил, то пользователь может сгруппировать их по уровням. Данное инструментальное средство поддерживает два способа определения трансформации: SPO, DPO.

Преимуществом данного подхода является то, что он был реализован в инструментальном средстве [32], которое содержит визуальные редакторы, интерпретатор, инструменты проверки допустимости, а также текстовый редактор описания Java-выражений.

Поскольку левая и правая части правила трансформации должны быть описаны в одной графической нотации (с помощью одного типизированного графа), то данный инструментарий может быть использован лишь для описания операционной семантики исходной модели, но не для трансформации модели из одной графической нотации в другую. Это является основным ограничением, которое не позволяет использовать данный инструментарий в системе MetaLanguage. Кроме того, в случае существования нескольких вхождений левой части правила в хост-граф, лишь для одного из них правило будет выполнено, хотя в MetaLanguage необходимо выполнить правило для всех вхождений.

Graph Rewriting and Transformation

GReAT (Graph REwriting And Transformation) – язык описания преобразований моделей, базирующийся на тройных графовых грамматиках [7]. Трансформация, описанная на этом языке, представляет собой набор упорядоченных правил переписывания графа, которые применяются к исходной модели и в результате создают целевую модель. Правило – это операция переписывания, представляющая собой паттерн, который необходимо найти в исходной модели, и подграф, который будет создан в результате применения правила. Правила всегда выполняются в некотором контексте, которым является исходный граф.

Данный язык использует модифицированный аппарат графовых грамматик. Грамматика по-прежнему состоит из набора правил, но правило уже не разбивается на левую и правую части, как это предполагается в классических графовых грамматиках. Вместо этого строится один граф, содержащий как левую, так и правую часть правила. Каждый элемент (вершина или ребро) может быть помечен одним из специальных символов, который указывает способ обработки этого элемента: «создание», «удаление» и др.

Язык GReAT включает в себя три языка: язык спецификации ме-

тамоделей, язык описания правил преобразования графа и язык описания потока управления.

Для спецификации исходной и целевой метамоделей используются диаграммы классов UML и ограничения, описанные на языке OCL. UML позволяет определять структурные ограничения, в то время как OCL может быть использован для задания неструктурных ограничений. Таким образом, диаграмма классов UML играет роль графовой грамматики, с помощью которой могут быть созданы метамодели предметных областей. Кроме того, описание метамоделей с помощью UML позволило разработчикам GReAT создать генератор объектно-ориентированного кода. Данный язык предоставляет пользователю возможность определять произвольное число метамоделей, которые могут использоваться в дальнейших преобразованиях.

Процесс создания трансформации в GReAT состоит из нескольких шагов. Сначала пользователю необходимо импортировать описание исходных и целевых метамоделей. Далее следует создать новую метамодель, содержащую все объекты исходных и целевых метамоделей, дополнительные вспомогательные вершины и ребра. После создания единой метамодели пользователь может приступить непосредственно к процессу описания правил преобразования.

Графы, являющиеся формальной основой GReAT, – *типизированные атрибутные мультиграфы*. Графы описываются с помощью диаграмм классов UML, в которых классы и ассоциации являются типами для вершин и рёбер графа, соответственно.

Вершина графа *v* представляет собой тройку (*class*, *id*, *attrs*), где *class* – класс UML, *id* – уникальный идентификатор вершины, *attrs* – отображение, которое каждому атрибуту класса ставит в соответствие некоторое значение.

Ребро e – это пятерка (*assoc*, *id*, *src*, *dst*, *attrs*), где *assoc* – ассоциация или класс ассоциации языка UML, *id* – уникальный идентификатор ребра, *src* и *dst* – инцидентные ребру вершины, *attrs* – отображение, которое каждому атрибуту ассоциации ставит в соответствие некоторое значение.

Граф *G* – это пара (*V*, *E*), где *V* – непустое множество вершин графа, *E* – множество рёбер графа, причём:

$$\forall e = (assoc, id, src, dst, attrs) \in E : src \in V \wedge dst \in V .$$

Соответствие – это пара (*MV*, *ME*), где *MV* – множество привязок вершин, *ME* – множество привязок ребер. Соответствие связывает элементы графа-паттерна и хост-графа. *Привязка вершины* – это пара (*pv*, *hv*), где *pv* – вершина графа-паттерна, *hv* – вершина хост-графа.

Привязка ребра – это пара (*pe*, *he*), где *pe* – ребро графа-паттерна, *he* – ребро хост-графа.

Следует отметить, что вершина графа-паттерна соответствует вершине хост-графа, если они имеют совместимый тип. Кроме того, соответствие является инъективным, т.е. один элемент графа-паттерна отображается ровно на один элемент хост-графа, однако, наборы соответствий не инъективны: один и тот же элемент хост-графа может присутствовать в нескольких соответствиях.

Основной объект преобразования в GReAT – *порождающее правило*, или *продукция*. Продукция содержит:

- граф паттерна, в котором каждый объект имеет свой тип: класс или ассоциация;
- роль;
- входной интерфейс – множество входных портов, на которые поступают объекты графа от предшествующих правил;
- выходной интерфейс – множество выходных портов, через которые объекты графа передаются следующим правилам;
- фильтр – выражение логического типа, которое определяет, должно ли правило быть выполнено для данного подграфа;
- отображение атрибутов – программный код, который выполняется для каждого допустимого соответствия, чтобы установить значения атрибутов вершин и рёбер графа.

Роль в продукции представляет собой отображение вершин и дуг паттерна на ограниченный набор действий, которые могут быть выполнены над ними. Выделяют три вида ролей:

- связывание: объект используется для сопоставления объектов в графе;
- удаление: объект используется для сопоставления объектов, но как только соответствие определено, объект удаляется;
- создание: после того, как соответствие определено, создаётся новый объект.

Если сопоставление прошло успешно, то для каждого соответствия объекты, имеющие роль «удаление» (в визуальном представление изображается в виде «крестика»), будут удалены, после чего объекты с ролью «создание» (в визуальном представление изображается в виде «галочки»), будут созданы. Операция удаления исключает вершину и все инцидентные ей ребра. Во время сопоставления может возникнуть ситуация, когда объект хост-графа уже удален из соответствия, в то время как следующее соответствие все еще содержит привязку для данного объекта, поэтому перед выполнением операции система проверяет существование объекта и, если он был уже удален,

сообщает о невозможности выполнения действия.

Иногда возникает ситуация, когда имеющихся паттернов недостаточно для задания точного соответствия. Так, например, необходимо указать, что целочисленный атрибут некоторой вершины должен находиться в пределах определенного диапазона. Существует также потребность инициализировать значения атрибутов созданных объектов и/или изменять атрибуты существующих объектов. Для этих целей авторы предлагают использовать C++ в качестве языка реализации фильтров и кода отображения атрибутов.

На рис. 2 представлено простое правило преобразования в GReAT [8]. Объекты «House» и «PurchaseOrder» имеют входной интерфейс, это означает, что они получают информацию от предыдущего правила, которое было выполнено. Эти объекты формируют начальный контекст правила для осуществления поиска паттерна, т.е. правило ищет объекты «Room1», «Room2» и «AdjacentTo» в объекте «House». Как только эти элементы найдены, система приступит к проверке фильтра «HasDoor», содержащего процедурный код на C++, позволяющий получить доступ к атрибутам объектов. Если фильтр вернул истинное значение, то будет создан новый объект «OrderItem» в связанном объекте «PurchaseOrder» и будет выполнен процедурный код отображения атрибутов, содержавшийся в блоке «AttributeMapping». В итоге, элементы «House» и «OrderItem» будут переданы следующему правилу через выходные интерфейсы.

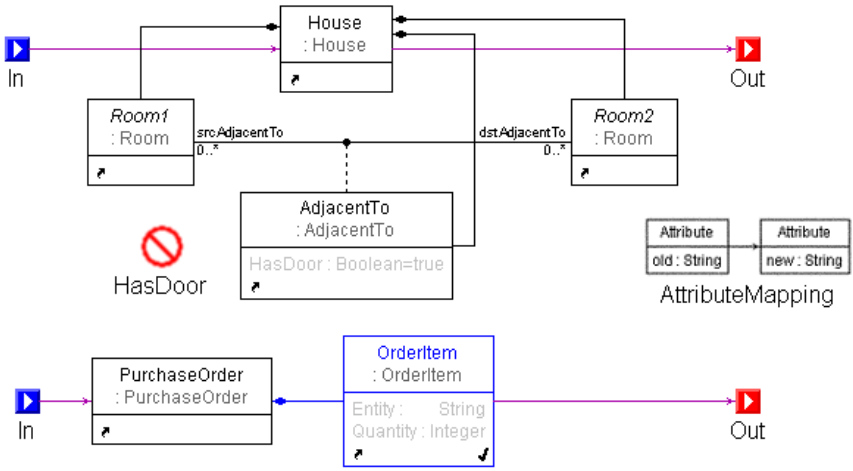


Рис. 2 Правило трансформации в GReAT

Важно отметить, что в GReAT и LHS, и RHS правила преобразования определяются вместе. Объекты, отмеченные как «связывание», можно рассматривать как левую часть правила, а объекты, отмеченные как «создание» или «удаление», – как правую часть правила.

В классических графовых грамматиках на очередном шаге применяется та продукция, которая может быть выполнена. Однако такой подход эффективен лишь для генерации языков, поскольку для модельных трансформаций часто появляется необходимость контроля последовательности срабатывания правил. Для этих целей в GReAT был встроен дополнительный компонент – язык потока управления, который позволяет управлять процессом преобразования моделей. Этот язык включает следующие конструкции потока управления [37]:

1. Упорядочение: если выходной интерфейс правила соединен с входным интерфейсом другого правила, то по окончании выполнения первого правила будет выполнено второе правило. Связь правил предполагает передачу потока пакетов от одного правила другому.
2. Недетерминизм: если правила не соединены последовательно, они могут быть выполнены параллельно, причем порядок срабатывания параллельных правил будет недетерминирован.
3. Иерархия: продукции могут иметь иерархическую структуру, которая создается с помощью конструкции Blocks. Blocks может содержать в себе правила и другие блоки. Есть два вида этой конструкции: Block и ForBlock. Единственное отличие их семантики в том, что каждый входной пакет в ForBlock проходит через все содержащие его правила прежде, чем следующий входной пакет будет подан на вход блоку, в то время как в Block все пакеты, обработанные первым правилом, передаются на вход следующему правилу.
4. Рекурсия: продукционное правило может вызывать себя.
5. Условное ветвление: GReAT предоставляет в распоряжение пользователя специальные блоки Test-Case для разветвления потока управления. Блок Test может иметь несколько блоков Case.

Последовательность выполнения правил завершается, если какое-либо правило не имеет выходного интерфейса, или если правило, имеющее выходной интерфейс, не производит выходных пакетов.

Подводя итог, можно говорить о том, что GReAT является языком описания трансформаций моделей, который использует диаграммы классов UML и язык OCL для представления преобразований. Данный язык позволяет производить трансформацию сразу для нескольких исходных метамоделей [6], что является значительным преимуществом

по сравнению с другими подходами. Язык преобразования состоит из трех подъязыков: языка спецификации метамоделей, языка описания правил трансформации графа и языка потока управления. Язык спецификации предоставляет возможность описывать достаточно сложные метамодели с помощью диаграмм классов UML. Язык описания правил трансформации используется для определения шагов преобразования графов. Язык потока управления позволяет управлять последовательностью срабатывания правил.

К преимуществам данного подхода следует отнести, во-первых, наличие возможности работы с несколькими исходными и целевыми метамоделями одновременно, что позволяет выполнять преобразование моделей, построенных с использованием нескольких языков моделирования; во-вторых, явное упорядочение продукционных правил позволяет управлять процессом их срабатывания, а использование операторов языка потока управления предоставляет возможность императивно описать процесс преобразования моделей.

Однако этот язык описания трансформаций обладает также и ограничениями. Так, в GReAT отсутствует возможность выбора языка спецификации метамоделей и изменения его описания, поэтому пользователю приходится довольствоваться возможностями, предоставляемыми UML и OCL. Кроме того, язык GReAT предъявляет высокие требования к уровню квалификации пользователя, поскольку фильтры описываются на языке OCL, а правила отображения атрибутов – на языке C++. Создание при описании трансформации единого домена, содержащего как левую, так и правую части правила, делает описание правил менее наглядным и более запутанным. Кроме того, отсутствует единая математическая основа данного подхода: так, в источниках [6] и [8] даны различные формальные определения графа, вершины, дуги.

Visual Automated Model Transformations

VIATRA (VIsual Automated model TRAnsformations) – основанный на правилах и паттернах фреймворк преобразования графовых моделей, который комбинирует в единую парадигму спецификации два формализма: первый основан на правилах трансформации графов [19] и используется для описания моделей, второй основан на абстрактных конечных автоматах и предназначен для описания потока управления.

Текущая версия реализации фреймворка [13] поддерживает текстовый язык преобразования VTL, который состоит из трех подъязыков: языка метамоделирования, языка описания трансформаций моделей и языка описания шаблонов кода. Язык метамоделирования (VPM) позволяет выполнять многоуровневое моделирование. Изначально для данного языка был создан лишь абстрактный синтаксис [36], однако в

дальнейшем разработчики предоставили в распоряжение пользователя два представления конкретного синтаксиса: текстовое (VTML) и визуальное. Текстовый формат удобен для автоматической генерации метамоделей, а графическая нотация, используемая для описания небольших метамоделей, является более наглядной для пользователей знакомых с языком MOF.

Язык VPM состоит из двух конструкций: сущность (обобщение пакета, класса и объекта в MOF) и отношение (обобщение ассоциации, атрибута и ссылки в MOF). Сущности описывают базовые понятия исходной и целевой предметной области, в то время как отношения описывают связи между этими понятиями.

Сущности метамодели представляют строгую иерархию включения, которая составляет пространство имен метамодели. В рамках сущности-контейнера каждый элемент метамодели имеет уникальное локальное имя, помимо этого, каждый элемент может также иметь глобальное уникальное имя.

Конструкция «отношение» имеет следующие свойства:

- свойство *isAggregation* указывает, что данное отношение является агрегацией;
- свойство *inverse* указывает на двунаправленный характер отношения;
- свойство *multiplicity* позволяет ввести ограничения на структуру модели.

Между сущностями метамодели могут быть установлены два вида специальных отношений:

- *supertypeOf* (наследование, обобщение) представляет отношение «суперкласс-подкласс» аналогично обобщению в UML;
- *instanceOf* представляет отношение «экземпляр-класс» между уровнями метамодели.

Рассмотрим пример описания метамодели с помощью языка VTML. Пусть дана упрощенная метамодель диаграмм классов языка UML (см. рис. 3). Ее описание на языке VTML будет следующим:

```
entity(UML)
{    entity(Class);
     entity(Association);
     entity(Attribute);
     relation(src, Association, Class);
     relation(dst, Association, Class);
     relation(parent, Class, Class);
     relation(attrs, Class, Attribute);
     multiplicity(attrs, many-to-many)
     relation(type, Attribute, Class);
}
```

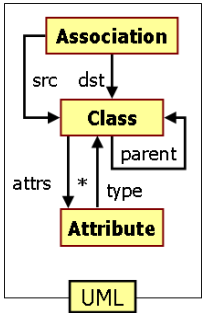


Рис. 3. Упрощенная метамодель диаграмм классов UML

Основные элементы VTML – декларативные конструкции, которые описываются по аналогии с фактами языка Prolog. Сущность должна быть объявлена в виде

Тип (*Имя*).

Если сущность не имеет конкретного типа, то она должна быть экземпляром системного типа *entity*. Поскольку сущности могут содержать в себе другие элементы модели, то разработчиками VIATRA была предусмотрена возможность описания вложенных блоков аналогично тому, как это определено в языке C, т.е. с помощью фигурных скобок ({}). Так, в рассматриваемом примере сущность-контейнер содержит в себе блок описания других объектов метамодели.

Отношение должно быть задано таким же образом, как и сущность, с той лишь разницей, что дополнительно необходимо указать исходную и целевую сущности. Синтаксис описания отношения следующий:

Тип (*Имя*, *Сущность-источник*, *Сущность-приемник*).

Особый вид отношений может быть представлен с помощью ключевых слов *supertypeOf* и *subtypeOf* для отношений специализации, и *typeOf* и *instanceOf* для отношений инстанцирования. Синтаксис задания этих отношений:

Отношение (*Класс*, *Экземпляр*).

Так, например, выражение `typeOf(UML.Class, Dog)` определяет, что сущность `Dog` является экземпляром элемента метамодели `UML.Class`. Благодаря использованию отношений такого вида, элементы модели могут иметь разные типы.

Для описания преобразований моделей VIATRA использует язык задания трансформаций моделей, основанный на паттернах преобразования. Паттерны графа задают условия, налагаемые на контекст моделей, правила преобразования определяют элементарные действия над моделью, а абстрактные конечные автоматы используются для описа-

ния потока управления. Язык, используемый для реализации всех этих операций, – это текстовый язык VTCL. В последующих версиях разработчики планируют реализовать графический редактор, который будет поддерживать возможность визуального описания преобразований моделей.

Паттерны графа – атомарные модули преобразования моделей. Они описывают контекст: условия или ограничения, которые должны быть соблюдены для выполнения определенных операций над моделями. Модель удовлетворяет паттерну графа, если между паттерном и подграфом модели может быть установлено соответствие на основе обобщённого метода сопоставления с паттерном, описанного в работе [34].

В следующем примере представлен паттерн, который может быть применен только для тех экземпляров классов, которые не имеют родительских классов:

```
pattern isTopClass(C) =
{    UML.Class(C);
     neg pattern negCondition(C) =
     {
          UML.Class(C);
          UML.Class.parent(C,CP);
          UML.Class(CP);
     }
     check (name(C)!="")
}
```

Паттерны задаются с помощью ключевого слова *pattern*, они могут иметь параметры, которые должны быть перечислены после имени паттерна, тело паттерна содержит определения сущностей и отношений модели, аналогично тому, как они описываются на языке VTML. Ключевое слово *neg* задает новый паттерн, который содержится в текущем и представляет собой отрицательное условие применения. Если для класса *C* существует родительский класс *CP*, то отрицательное условие будет истинным и сопоставление с паттерном будет неуспешным.

В VTCL паттерн может быть вызван другим паттерном с помощью конструкции *find*, благодаря этому поддерживается возможность повторного использования существующих паттернов.

Трансформации моделей задаются правилами преобразования графа, которые используют паттерны для определения критериев применимости преобразования. При выполнении правила трансформации подграф, представленный в левой части правила (LHS), будет заменен паттерном, представленным в правой части правила (RHS).

Язык VTCL поддерживает два способа задания правил трансформации графа. Первый – традиционная нотация, содержащая два паттерна: паттерн предусловия для LHS и паттерн постусловия, который определяет RHS правила. Элементы, которые присутствуют в LHS, в результате применения правила будут заменены на элементы RHS, а остальные элементы модели останутся неизменными. Такие правила преобразования графов задаются с помощью ключевого слова *gtrule* и могут иметь направленные параметры (in/out/inout). Обмен информацией между паттернами одного правила происходит через передачу параметров.

Второй способ задания правил преобразования в VIATRA – графическая нотация, заимствованная из подхода Fujaba [27]. Правило содержит простой паттерн, который определяет контекст применения правила преобразования графа, и действия, которые должны быть выполнены. Элементы паттерна, отмеченные с помощью ключевого слова *new*, создаются после того, как соответствие для LHS было установлено, а элементы, отмеченные ключевым словом *del*, удаляются после выполнения сопоставления.

Для повторного использования алгоритмов преобразования, не зависимых от конкретной метамодели, VIATRA поддерживает универсальные метапреобразования, которые основаны на применении многоуровневого моделирования.

Управление порядком и режимом выполнения правил преобразования осуществляется командами языка ASM, которые имеют формальную семантику и аналогичны конструкциям языков программирования. Основные элементы программы, описанной на ASM, – правила, аналог методов объектно-ориентированных языков, переменные и функции. Помимо стандартных математических функций, VIATRA предоставляет возможность определять собственные функции, которые пользователь может описать на языке Java. Это предоставляет возможность реализовать сложные вычисления, которые могут использоваться при выполнении преобразований моделей.

Для поддержки трансформации вида «модель-текст» язык VTCL включает конструкцию *print*, которая позволяет получить исходный код всех преобразований на языке Java. Этот код может быть отформатирован с помощью специальных операторов форматирования кода. Для комбинирования в одном файле блоков статичного текста и динамических данных используется понятие шаблона кода. Шаблон – текстовый блок, который может включать ссылки на переменные ASM и какие-либо конструкции.

Подводя итог, можно говорить о том, что VIATRA является под-

ходом к описанию преобразований моделей, который интегрирует в себе графовые трансформации и абстрактные конечные автоматы. Благодаря использованию конструкций ASM разработчикам фреймворка удалось значительно повысить семантику стандартных языков описания паттернов и преобразования графов.

К преимуществам VIATRA следует отнести, во-первых, возможность использования универсальных метапреобразований, что позволяет производить многоуровневое моделирование и повторное использование уже созданных алгоритмов трансформаций. Во-вторых, основанный на шаблонах метод генерации кода для преобразований вида «модель-текст» позволяет генерировать файлы, содержащие как статичные части, так и изменяемые данные. В-третьих, возможность использования как текстовой, так и графической нотации языка метамоделирования предоставляет пользователю возможность выбирать наиболее удобное для него представление языка.

Недостатком подхода является невыразительность и сложность текстового языка описания метамоделей и трансформаций. Хотя разработчики фреймворка критикуют стандарт MOF за отсутствие возможности многоуровневого моделирования [9], они все же остаются в рамках той же парадигмы при использовании визуального языка описания метамоделей, который используется чаще, чем текстовое представление. Данный инструментарий предъявляет высокие требования к уровню квалификации конечного пользователя, поскольку для описания трансформаций пользователь должен не только знать язык метамоделирования MOF, но и уметь строить абстрактные конечные автоматы. Кроме того, у пользователя VIATRA отсутствует возможность выбора языка спецификации метамоделей и трансформаций, так как для этих целей может быть использован только язык MOF.

Query/View/Transformation

QVT (Query/View/Transformation) – стандарт трансформации моделей, предложенный OMG.

Стандарт QVT предоставляет в распоряжение пользователя несколько языков преобразования моделей, все они работают с моделями, метамодели которых описаны с помощью стандарта MOF. QVT для описания преобразований использует язык OCL, расширенный императивными функциями. Ключевыми понятиями этого стандарта являются: запрос, представление, трансформация [24].

Запрос – некоторое выражение, применяемое к модели, результатом выполнения которого является один или несколько объектов этой модели. Формулировка запроса с помощью QVT аналогична построению запросов на языке SQL к реляционной базе данных (БД). Так,

например, запрос к модели диаграмм классов языка UML может быть следующим: «Выбрать все объекты, которые не имеют родительского класса». Чаще всего запросы в QVT используются для задания фильтров. Запросы также могут быть созданы с помощью операционной семантики [33].

Представление – модель, которая получена из другой (основной) модели. Представление не может быть изменено независимо от модели, из которой оно получено. Изменения в основной модели приводят к соответствующим изменениям в представлении. В общем случае метамодель представления отлична от исходной метамодели.

Представления обычно не сохраняются независимо от их исходных моделей и часто доступны только для чтения. Если представление доступно для редактирования, то все произведенные в нем изменения будут отражены в основной модели, поэтому для редактирования представления необходимо иметь определенное его отображение на основную модель. Представление может быть частичным, т.е. основанным на некотором подмножестве исходной модели, или полным, т.е. охватывающим тот же объем информации, что и исходная модель, но реорганизованный для конкретной задачи или пользователя. Следует отметить, что запрос является частным случаем представления.

Трансформация – преобразование, позволяющее получить целевую модель из исходной. Результатом трансформации может быть независимая или зависимая модель. В первом случае нет никакой связи между исходной и целевой моделью с момента, как только последняя была сгенерирована. Во втором случае преобразование связывает исходную и целевую модели.

Преобразования могут быть однонаправленными и двунаправленными. В случае двунаправленных трансформаций пользователь может модифицировать как исходную, так и целевую модель, при этом изменения будут распространены в любом направлении. Если изменения были произведены одновременно в двух моделях, то появляются конфликтные ситуации, которые автоматически разрешены быть не могут, – в этом случае необходимо вмешательство пользователя.

Представление является частным случаем трансформации, в которой целевая модель не может быть изменена независимо от исходной модели. Если представление доступно для редактирования, то соответствующее преобразование должно быть двунаправленным, чтобы существовала возможность изменения исходной модели. Таким образом, базовым элементом стандарта QVT является трансформация, состоящая из набора правил.

Правила – это модули, из которых состоят трансформации. Пра-

вило ответственно за преобразование определенного подмножества элементов исходной модели в соответствующие элементы целевой модели. Правило может содержать описание и/или реализацию. Чисто декларативное правило будет содержать только описание, чисто императивное правило – только реализацию, гибридное правило содержит как декларативные, так и императивные элементы.

Описание является спецификацией отображения между элементами исходной и целевой модели. Описание может содержать достаточно информации для полного определения однонаправленного или двунаправленного преобразования. Альтернативой является использование описания только для определения контекста исходной и целевой модели, в то время как для выполнения самой трансформации вызывается реализация.

Реализация – императивная спецификация действий, позволяющих явно создавать элементы целевой модели из элементов исходной модели. Реализации, как правило, направлены, т.е. они выполняются слева направо или справа налево, однако существует возможность создания реализаций, которые могут работать в любом направлении.

Еще одним понятием QVT является *соответствие*, оно устанавливается во время применения преобразования, если элементы левой и/или правой модели удовлетворяют условиям фильтра правила. Соответствие инициирует создание элементов целевой модели, управляемой описанием и/или реализацией соответствующего правила.

Взаимосвязь между декларативным и императивным уровнем спецификации QVT представлена на рис. 4 [17].

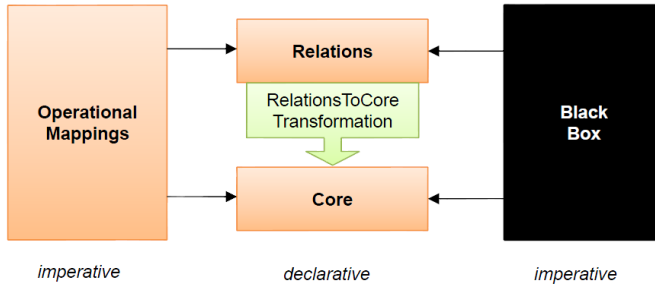


Рис. 4. Связь между языками QVT

Как видно из рисунка, QVT содержит два декларативных языка описания преобразований, которые формируют два уровня абстракции. Есть более абстрактный и более удобный для пользователя *язык отношений* (*Relations*), конструкции которого отображаются на более

конкретный *язык ядра* QVT (Core). В обоих языках для описания паттернов моделей используются выражения OCL.

Язык отношений – декларативный язык, позволяющий описывать однонаправленные и двунаправленные трансформации моделей. Для проверки трансформации на непротиворечивость используется режим выполнения *checkonly*; если набор правил трансформации является непротиворечивым, то система вернет результат «истина», иначе – «ложь». Язык отношений поддерживает сопоставление с паттерном составного объекта и неявно создает классы трассировки и их экземпляры для записи действий, выполненных во время преобразования.

Мощности языка отношений и языка ядра одинаковы, хотя семантика последнего является более простой. В языке ядра модели трассировки определяются явно, а не выводятся из описания преобразований, как в языке отношений. Пользователь может описывать трансформацию непосредственно на языке ядра или на языке отношений, конструкции которого будут в дальнейшем отображены на элементы ядра. Следует отметить, что язык ядра не имеет визуального синтаксиса.

Рассмотрим пример. Опишем с помощью языка отношений двунаправленное преобразование метамодели диаграмм классов UML в метамодель реляционной БД [29]:

```
transformation umlToRdbms
      (uml:SimpleUML, rdbms: SimpleRDBMS)
{  relation ClassToTable
   {  domain uml c:Class
      {  namespace = p:Package {},
         kind='Persistent',
         name=cn
      }
      domain rdbms t:Table
      {  schema = s:Schema {},
         name='t_' + cn,
         column = cl:Column
         {name=cn+'_tid',
           type='NUMBER'
         },
         primaryKey = k:PrimaryKey
         {name=cn+'_pk'
           column=cl
         }
      }
      when { PackageToSchema(p, s);}
      where { AttributeToColumn(c, t);}
   }
}
```

Описание преобразования начинается с ключевого слова *transformation*, за которым следует имя трансформации. Далее следует

описание метамоделей диаграмм классов и реляционной БД, каждое из которых начинается с ключевого слова *domain*. По окончании описания метамоделей необходимо определить связь между ними с помощью разделов *when* и *where*.

В дополнение к декларативным языкам, которые реализуют одну семантику на двух разных уровнях абстракции, QVT поддерживает два способа описания императивных конструкций: стандартный язык – операционные отображения (Operational Mappings), а также нестандартные реализации операций из MOF – чёрный ящик (Black Box).

Язык операционных отображений является стандартным способом описания императивных преобразований моделей. Этот язык использует конструкции OCL, расширенные возможностями процедурного стиля и конкретным синтаксисом. Операционные отображения могут использоваться для реализации одного или нескольких отображений, если их сложно описать с помощью лишь декларативных спецификаций. Преобразования, полностью описанные на языке операционных отображений, называются операционными преобразованиями.

Существование в стандарте QVT *черного ящика* позволяет использовать в процессе задания трансформаций средства преобразования, описанные на других языках, например XSLT или XQuery. Благодаря этому появляется возможность создавать сложные алгоритмы преобразования на языках высокого уровня с привязкой к MOF. Кроме того, такой подход позволяет использовать специализированные библиотеки для вычисления значений свойств элементов модели, например библиотеки с математическими функциями.

Одной из модификаций стандарта QVT являются подходы, описанные в работе [1]. Трансформация представляет собой последовательность правил, применяемых к конкретным элементам модели или совокупности элементов. Правило состоит из *секции выборки*, описывающей контекст правила, и *секции генерации*, определяющей само преобразование. Секция выборки состоит из операторов выборки и уточняющих условий. Секция генерации содержит операторы создания, модификации, удаления элементов модели. При выполнении трансформации создается *трансформационная связь*, которая содержит служебную информацию о сработавших правилах и выполненных операциях создания, модификации, удаления элементов модели. Трансформационная связь используется для поддержания исходной и целевой моделей в согласованном состоянии. Предложенные в работе подходы применяются для описания и выполнения трансформаций моделей, созданных в нотации UML.

Таким образом, QVT – предложенный OMG стандарт трансфор-

мации моделей, предоставляющий в распоряжение пользователя как декларативные, так и императивные языки. Преобразование определяется на уровне метамоделей, описанных с помощью MOF. Достоинством данного подхода является существование стандарта его описания, а также использование в процессе построения преобразований моделей стандартных языков: OCL и MOF. Еще одним преимуществом QVT является широкий набор языков описания трансформаций, которые позволяют использовать как стандартные средства, так и их расширения.

Однако эти преимущества имеют и обратную сторону. Использование MOF в качестве языка метамоделирования не позволяет пользователю выбрать удобный для него метаязык, а также изменить описание метаязыка интегрированного в QVT. Кроме того, данный стандарт не позволяет производить трансформации вида «модель-текст», так как каждая метамодель должна быть описана с помощью стандарта MOF.

Для использования в процессе определения трансформаций внешних преобразований в виде черного ящика необходимо не только иметь хорошо отлаженные библиотеки функций, но и запретить модификацию элементов моделей из этих библиотек напрямую.

Как видно из приведенного примера описания трансформации с помощью QVT, синтаксис языка преобразования моделей достаточно сложен по сравнению с другими подходами. Хотя разработчики стандарта заявляют о поддержке как визуального, так и текстового представления языка описания трансформаций, в большинстве работ авторы предпочитают использовать лишь текстовый конкретный синтаксис, поскольку визуальное представление оказывается еще более сложным для понимания.

Следует также отметить, что декларативный язык ядра никогда не имел полной реализации, поэтому в настоящее время нет ни одного инструментального средства, поддерживающего весь стандарт QVT.

ATLAS Transformation Language

Язык трансформации ATL (ATLAS Transformation Language) [20] базируется на стандарте QVT. ATL разработан как часть платформы AMMA (ATLAS Model Management Architecture) [10]. *Абстрактным* синтаксисом ATL является язык MOF, а в качестве *конкретного* синтаксиса используется текстовый гибридный язык, который интегрирует в себе как декларативные, так и императивные конструкции. Дополнительно существует графический синтаксис, который позволяет частично определить правила трансформации моделей.

По сравнению с QVT, язык ATL имеет *ограничение на создаваемые правила*: он позволяет описывать лишь такие правила, в левой ча-

сти которых находится только один элемент модели – вершина или дуга. Это дает возможность автоматически проверять правильность описанной трансформации, хотя и сильно сужает выразительные возможности языка и область его применения.

Трансформации в ATL описываются в специальных файлах-модулях [21]. *Модуль* содержит обязательный раздел *заголовка*, несколько *помощников* и *правила преобразования*. В разделе заголовка указывается имя модуля преобразования, и объявляются исходные и целевые метамодели. Помощники и правила преобразования – конструкции, используемые для определения функциональной составляющей трансформации.

Помощники используются для навигации по элементам исходной модели и ее атрибутам. Термин «помощник» заимствован из спецификации языка OCL, который содержит два вида помощников: *помощник операции* и *помощник атрибута*. В ATL помощник может быть определен только для OCL-элементов или объектов исходной модели. Основная цель помощника операции – выполнение навигации по исходной модели. Помощники атрибута позволяют производить навигацию по атрибутам элементов исходной модели. Как и у помощников операции, у них есть имя, контекст и тип, отличие лишь в том, что они не могут иметь входных параметров. Помощники атрибута могут использоваться для установления связи между исходными моделями, так как тип помощника атрибута может быть классом метамодели, отличной от исходной метамодели. Существует возможность рекурсивного определения помощников атрибута.

Основной конструкцией языка ATL, используемой для выражения логики трансформации, является *правило преобразования*. Преобразование определяется на уровне метамоделей. Правило преобразования может быть вызвано явно с помощью его имени (императивное правило) или быть выполнено в результате обнаружения соответствия – вхождения паттерна в исходную модель (декларативное правило).

Декларативные правила состоят из двух элементов: исходный и целевой паттерн.

Исходный паттерн представляет собой объединение множества типов, полученных из исходной метамодели, и типов, доступных в языке OCL, с множеством фильтров, налагающих ограничения на исходную модель. *Фильтр* представляет собой логическое выражение и используется для выбора элементов исходной метамодели, удовлетворяющих определенным условиям, описанным на языке OCL. Таким образом, исходный паттерн – это множество элементов исходной метамодели, удовлетворяющих условиям фильтров.

Целевой паттерн состоит из множества типов, полученных из целевой метамодели, и привязок. Привязка определяет выражение, значение которого используется для инициализации некоторого элемента (атрибута, отношения или роли ассоциации). Таким образом, целевой паттерн – множество вершин целевой метамодели, которые могут быть соединены привязкой.

Рассмотрим пример. Опишем правило преобразования понятия «Класс», содержащего один атрибут «Имя», в понятие «Таблица» [21]:

```
rule Класс2Таблица
{
     from
          c: Класс (c.parent.oclIsUndefined())
     to
          t: Таблица (Имя <- c.Имя)
}
```

Как видно из примера, имя правила задается после ключевого слова *rule*. Исходный паттерн правила начинается с ключевого слова *from* и содержит определение переменной типа «Класс». Фильтр задает условие преобразования: только к классам без суперклассов правило может быть применено. Целевой паттерн начинается с ключевого слова *to*, в этом паттерне определены переменная типа «Таблица» и привязка, которая содержит выражение, используемое для инициализации атрибута «Имя».

Более подробное описание примера задания трансформации на языке ATL представлено в работе [4].

По *способу вызова* все правила можно разделить на

- *Стандартные правила*, которые вызываются один раз для каждого соответствия, найденного в исходной модели.
- *Ленивые правила*, вызываемые другими правилами. Они вызываются столько раз, сколько это указано в содержащих их правилах. Это означает, что ленивое правило может быть применено многократно в одном соответствии, каждый раз создавая новый набор целевых элементов.
- *Уникальные ленивые правила*, которые инициируются другими правилами. Они могут быть вызваны только один раз для данного соответствия. Если уникальное ленивое правило вызывается повторно в том же соответствии, то вместо создания новых элементов модели будут использоваться уже существующие.

У декларативного стиля спецификации трансформации есть много преимуществ. Он основан на определении отношений между исходны-

ми и целевыми паттернами и соответствует интуитивному определению трансформации. Этот стиль скрывает детали, связанные с выбором исходных элементов, срабатыванием правил, их упорядочиванием и т.д., поэтому он позволяет скрыть сложность алгоритмов трансформации за простотой синтаксиса языка.

Если в программе ATL используется вызываемое правило или блок операций, то эта программа не является полностью декларативной.

В ATL существует возможность наследования правил, которая может быть применена для повторного использования кода и/или задания полиморфных правил. Также существует возможность определить абстрактное правило, которое не может быть выполнено, но используется в качестве родительского.

Рассмотрим алгоритм выполнения правил трансформации моделей [11]. В первую очередь выполняется правило, помеченное как точка входа (entrypoint). Тело этого правила может содержать произвольное число вызываемых правил соответствия.

При первом вызове каждого правила соответствия создаются и инициализируются элементы целевой модели. Инициализация происходит на основе привязки. Все паттерны каждого правила сопоставляются с исходной моделью, и проверяются фильтры. Если возвращаемое фильтром значение – истина, то паттерн был распознан и правило может быть применено к некоторому подмножеству элементов исходной модели.

В результате выполнения правила преобразования, помимо элементов целевой модели, дополнительно создается ссылка трассировки. Эта ссылка связывает три компонента: правило, исходные и целевые элементы.

Для выполнения правил в ATL используется специальный алгоритм, получивший название алгоритма разрешения. Прежде чем присвоить значение привязки целевой функции, оно должно быть разрешено. Разрешение значения зависит от его типа. Если тип простой, то значение будет просто присвоено соответствующей функции. Если тип значения – класс метамодели, то возможно два варианта:

- значение привязки – целевой элемент, тогда оно будет просто присвоено этому элементу;
- значение привязки – элемент исходной модели, тогда сначала он должен быть преобразован в целевой элемент на основе ссылки трассировки, затем значение должно быть записано в соответствующий элемент целевой модели.

По окончании выполнения всех декларативных правил должно

быть выполнено правило, помеченное как точка выхода (endpoint).

Преобразования в ATL могут быть описаны с помощью конкретного графического синтаксиса, однако визуальный эквивалент есть не для всех конструкций языка, а только для базовых. Так, ни фильтры исходных паттернов, ни привязки целевых паттернов не могут быть представлены графически. Основное назначение этого синтаксиса состоит в том, чтобы сделать использование графических конструкций удобным и наглядным для определения паттернов декларативных правил. Этот синтаксис может использоваться для облегчения понимания преобразования.

Таким образом, ATL является языком описания трансформаций, позволяющим задавать преобразования любой исходной модели в указанную целевую модель. Преобразование производится на уровне метамоделей. Интеграция в рамках одного языка декларативного и императивного подхода позволяет производить преобразования для сложных предметных областей, при этом декларативный подход позволяет скрыть сложность алгоритмов трансформации.

В основе ATL лежит стандартизованный язык описания ограничений OCL, что позволяет использовать в процессе описания трансформаций все преимущества данного языка: стандартные типы, фильтры, помощники и др.

К ограничениям данного языка следует отнести высокие требования, предъявляемые к уровню квалификации разработчика преобразования. Поскольку ATL в большинстве случаев использует лишь текстовое определение трансформации, то, помимо знания исходной и целевой метамоделей, разработчик должен знать сам язык преобразования и язык описания ограничений OCL.

Кроме того, использование императивных конструкций элиминирует преимущества декларативного подхода. Отсутствие навигации по целевой модели затрудняет процесс определения обратных правил преобразования.

Использование языка MOF для спецификации метамоделей ограничивает возможность выбора метаязыка и изменения его описания. В силу этих же причин в ATL отсутствует возможность многоуровневого моделирования.

Трансформация моделей по образцу

Подход к трансформации моделей по образцу (Model Transformation By-Example, MTBE) основан на подходах программирования по образцу и формирования запросов по образцу.

Программирование по образцу [14] позволяет автоматизировать работу конечного пользователя, которая требует навыков разработки программ и знания высокоуровневых языков программирования за счет записи действий пользователя, например с помощью моделей трассировки, и генерации на основе этих моделей исходного кода программы. Это подобно тому, как Microsoft Office позволяет произвести запись взаимодействия пользователя с программным интерфейсом и сгенерировать код для всех произведенных пользователем операций. Полученный таким образом макрос может использоваться для автоматического воспроизведения записанных действий.

Цель формирования запросов по образцу [39] состоит в том, чтобы облегчить работу пользователя при построении запросов к реляционной БД за счет их автоматической генерации. Это достигается путем использования шаблонов таблиц, состоящих из образцов кортежей, заполненных некоторыми константами, ограничениями и командами. Команды описывают операции, которые необходимо произвести над данными (такие как выборка, вставка, удаление кортежей и др.). На основе шаблонов таблиц строятся запросы на языке SQL, которые могут быть выполнены во время работы с БД.

Основная цель MTBE – автоматическая генерация правил трансформации на основе начального набора образцов. Исходные образцы представляются тройками (*SMD, TMD, MB*), где *SMD* – исходная модель, *TMD* – целевая модель, *MB* – множество отображений, которое каждому элементу из *SMD* ставит в соответствие эквивалентный элемент из *TMD*. Начальный набор образцов определяет различные варианты преобразования моделей. Таким образом, пользователю достаточно описать преобразование на уровне моделей, не касаясь особенностей метамоделей.

Основное преимущество MTBE по сравнению с рассмотренными ранее подходами состоит в том, что при выполнении трансформации используются понятия исходного и целевого языков моделирования для спецификации моделей, в то время как правила трансформации автоматически генерируются на некотором языке, например на ATL [38] или на языке трансформации графов [35].

Процесс автоматической генерации правил трансформации моделей состоит из следующих шагов:

Шаг 1: Построение начального набора образцов. Разработчик преобразования составляет начальный набор взаимосвязанных исходных и целевых пар моделей. Это может быть как единственная модель, которая будет содержать множество всех концептов языка, так и несколько моделей, каждая из которых сосредотачивается на некотором

определенном концепте. Построенные модели, во-первых, должны соответствовать метамоделям языков, с помощью которых они созданы; во-вторых, они должны покрывать множество всех конструкций обоих языков моделирования.

Шаг 2: Автоматическая генерация правил трансформации. MTBE-система создает правила, которые максимально точно преобразуют множество исходных моделей в их целевые эквиваленты. Различные реализации используют для описания правил трансформации различные языки.

Шаг 3: Усовершенствование правил преобразования моделей. После автоматической генерации правил преобразования может потребоваться некоторая дополнительная информация от пользователя для того, чтобы разрешить возникшие конфликты отображения. Разработчик преобразования может модифицировать правила, добавляя дополнительные условия и обобщая существующие правила.

Шаг 4: Автоматическое выполнение правил. Разработчик проверяет корректность сгенерированных правил, выполняя их на тестовых парах моделей. Если результат трансформации неудовлетворителен, то процесс разработки правил может быть запущен вновь. В качестве тестовых моделей авторы работы [38] предлагает использовать исходный набор образцов. В других работах при тестировании предлагается использовать наборы, отличные от исходного.

В работе [35] авторы описывают полуавтоматический процесс генерации правил трансформации моделей, использующий индуктивное логическое программирование [26]. Начальный набор образцов на внутреннем уровне представляется на языке исчисления предикатов первого порядка, что значительно упрощает получение правил преобразования на языке трансформации графов. Для получения результирующего набора правил преобразования часто требуется усовершенствование сгенерированных правил «вручную».

Еще одна реализация MTBE описана в работе [22]. В ней авторы рассматривают проблему трансформации при отсутствии исчерпывающего набора образцов. Авторы сводят проблему трансформации моделей к задаче оптимизации методом роя частиц [28]. Процесс получения правил преобразования становится поиском решения в многомерном пространстве, где каждая размерность представляет определенный элемент модели, а каждая точка в пространстве поиска – возможный вариант трансформации этого элемента.

Поскольку правила трансформации модели заранее не известны, то вариантов преобразования конструкции может быть несколько, степень применимости каждого из которых характеризуется *качеством*

преобразования. Качество преобразования зависит от двух критериев: адекватности преобразования и когерентности преобразования относительно трансформации остальных конструкций. Качество преобразования исходной модели – сумма качеств преобразований каждого из ее элементов, следовательно, поиск оптимального преобразования эквивалентен поиску комбинации преобразований отдельных элементов модели. Однако пространство поиска может быть достаточно большим, если число элементов велико. Эвристический поиск методом роя частиц позволяет сократить это пространство.

Такой подход предоставляет целый ряд преимуществ. Во-первых, для любой исходной модели трансформация производится без генерации правил преобразования. Во-вторых, этот подход не зависит от способа представления исходной и целевой модели. В-третьих, подход не требует никакой дополнительной информации, кроме исходного набора образцов.

Чтобы еще более упростить процесс генерации правил трансформаций моделей авторы работы [31] предлагают новый подход – преобразование модели на основе демонстрации (MTBD). Вместо идеи MTBE получить правила преобразования из начального набора образцов, пользователей просят продемонстрировать, как преобразование модели должно быть выполнено с помощью определенного набора команд, например добавление, удаление элементов, их соединение и др. Эта демонстрация является основой для анализа трансформации моделей и получения паттерна преобразования. Система запоминает все произведенные пользователем операции. Последовательность записанных операций указывает, как должно быть выполнено преобразование. Однако не все операции значимы, так, например, возможно, что пользователь сначала создает некоторый элемент, модифицирует его, а затем понимает, что он является лишним, и удаляет, поэтому после записи демонстрации автоматически выполняется процесс оптимизации, который устраняет все лишние операции.

Данный подход не предполагает генерации правил преобразования моделей, вместо этого он строит паттерн преобразования, описывающий контекст преобразования, т.е. условия его применения, и содержание преобразования, т.е. набор действий, которые должны быть выполнены. После того, как паттерн был получен, система сохраняет его в БД. В процессе работы системы он может быть применен к любой модели, удовлетворяющей контексту.

Таким образом, MTBD позволяет задавать преобразования без необходимости использования какого-либо языка трансформации моделей. Кроме того, описание преобразования не требует от конечного

пользователя знания метамоделей.

Однако этот подход для эффективной работы требует большого количества демонстраций, полученных от пользователя, и его активного участия. Фактически, пользователь должен выбрать подходящий паттерн преобразования. Кроме того, авторы в своей работе не показывают, как именно MTBD может выполнить трансформацию всей исходной модели, а ограничиваются лишь применением паттернов к определенным ее фрагментам. Еще одним существенным ограничением данного подхода является то, что он может быть использован лишь для эндогенных трансформаций.

Таким образом, можно говорить о том, что трансформация модели по образцу является новым подходом, который пытается устранить ряд ограничений традиционных подходов к описанию преобразований моделей, связанных с явным заданием пользователем правил трансформации. Вместо того чтобы описывать правила преобразования «вручную», MTBE позволяет пользователю определить набор взаимосвязанных отображений между исходными и целевыми моделями, а правила преобразования на уровне метамоделей будут сгенерированы автоматически.

Основное преимущество MTBE заключается в том, что для получения правил преобразования пользователю не требуется знание какого-либо языка трансформации моделей, будь то графовые трансформации, ATL и др.

Однако ни один из упомянутых подходов не гарантирует того, что сгенерированные правила преобразования моделей корректны или полны. Более того, полученные правила сильно зависят от исходного набора образцов. Одним из требований, предъявляемых к образцам, является то, что они должны покрывать все множество конструкций языка, однако это нетривиальная задача, особенно в тех случаях, когда язык моделирования достаточно мощный. Также следует отметить, что поскольку правила преобразования генерируются на основе исходного множества пар моделей, то даже небольшие ошибки в определении отображения одной модели на другую могут привести к существенным недочетам в сгенерированных правилах трансформации.

MTBE является итеративным и интерактивным процессом. На практике получить заключительный набор правил трансформации на первой итерации практически невозможно. Это означает, что после автоматической генерации правил преобразования из начального набора образцов, эти образцы должны быть скорректированы или правила преобразования изменены, если пользователь не удовлетворен полученным результатом. Однако в большинстве случаев решение о том,

требуется ли модификация образцов и/или правил, является неочевидным для конечного пользователя – непрофессионального ИТ-специалиста.

Текущие реализации подхода MTBE позволяют выполнять лишь полные эквивалентные отображения, не учитывая сложные преобразования атрибутов, хотя на практике достаточно часто требуется провести преобразование атрибута некоторого элемента исходной модели в соответствующий атрибут элемента целевой модели, выполнив над ними некоторые арифметические или строковые операции, которые описаны на некотором языке трансформации.

Еще одним существенным ограничением большинства реализации MTBE является то, что в них не затрагивается проблема трансформации ограничений, налагаемых на элементы модели.

Выводы

Выше были рассмотрены различные языки и инструментальные средства описания трансформаций визуальных моделей. По результатам анализа можно сделать следующие выводы.

Различные модификации алгебраического подхода используются в AGG, GReAT, VIATRA. В AGG в качестве левой и правой части продукционного правила выступают типизированные атрибутные графы, причем обе части правила должны быть описаны в одной графической нотации, т.е. эта система позволяет выполнять лишь эндогенные преобразования, что делает невозможным ее применение в системе MetaLanguage. Кроме того, данный инструментарий не позволяет производить трансформации вида «модель-текст». Однако использование в качестве формальной основы алгебраического подхода к трансформации графов позволяет произвести парсинг графа, проверить графовую модель на непротиворечивость, а расширение графов возможностями языка Java делает трансформации достаточно мощными с функциональной точки зрения.

Язык GReAT основывается на алгебраическом подходе с двойным выталкиванием, поэтому для описания трансформации необходимо создать домен, содержащий как левую, так и правую часть продукционного правила одновременно с указанием того, какие элементы следует создать, а какие удалить. Такой вид правила является непривычным для конечного пользователя и немного запутанным. Однако такое правило предоставляет возможность выполнять трансформацию сразу для нескольких исходных метамоделей, что является значительным преимуществом по сравнению с другими подходами. Для описания метамоделей GReAT использует UML и OCL, что не позволяет пользо-

вателю выбирать язык спецификации метамоделей, изменять его описание. Это ограничивает применение данного подхода в системе MetaLanguage.

VIATRA – основанный на правилах и паттернах фреймворк, который комбинирует в единую парадигму спецификации два подхода: алгебраический подход для построения моделей и абстрактные конечные автоматы, предназначенные для описания потока управления. Благодаря использованию конструкций конечных автоматов, разработчикам удалось значительно повысить семантику стандартных языков описания паттернов и преобразований графов.

Одним из ограничений VIATRA является невыразительность текстового языка описания метамоделей. Так, для задания отношения между сущностями пользователю необходимо вновь определить сами сущности. Хотя разработчики фреймворка критикуют стандарт MOF за отсутствие возможности многоуровневого моделирования, они все же остаются в рамках той же парадигмы при использовании визуального языка описания метамоделей, который применяется чаще, чем текстовое представление, особенно пользователями, не являющимися профессиональными разработчиками.

Фреймворк VIATRA не предназначен для выполнения горизонтальных трансформаций моделей. Его основное назначение – верификация и валидация построенных моделей путем их трансформации. Стандарт QVT оказался более подходящим для этих целей, поэтому авторы смогли разработать простой и надежный алгоритм верификации. В результате, они отказались от наглядности в пользу надёжности.

QVT – предложенный OMG стандарт трансформации моделей, предоставляющий в распоряжение пользователя как декларативные, так и императивные языки. Преобразование определяется на уровне метамоделей, описанных с помощью MOF. Достоинством данного подхода является существование стандарта его описания, а также использование стандартных языков OCL и MOF в процессе построения преобразований моделей. Еще одним преимуществом QVT является наличие нескольких языков описания трансформаций, которые позволяют использовать как стандартные средства, так и их расширения.

Однако эти преимущества имеют и обратную сторону. Использование MOF в качестве языка метамоделирования не позволяет пользователю выбрать удобный для него метаязык, а также изменить описание метаязыка интегрированного в QVT. Кроме того, данный стандарт не позволяет производить трансформации вида «модель-текст», так как каждая метамодель должна быть описана с помощью MOF. Все это накладывает ряд ограничений на возможность применения стандарта

QVT в системе MetaLanguage.

Синтаксис текстового языка описания трансформаций в QVT достаточно сложен по сравнению с другими подходами. Хотя разработчики стандарта заявляют о поддержке как визуального, так и текстового представления языка описания трансформаций, в большинстве работ авторы предпочитают использовать лишь текстовый язык, поскольку визуальное представление оказывается еще более сложным для понимания.

Следует также отметить, что декларативный язык ядра никогда не имел полной реализации, поэтому в настоящее время нет ни одного инструментального средства, поддерживающего весь стандарт QVT.

ATL – язык, позволяющий описывать трансформации любой исходной модели в указанную целевую модель. Преобразование производится на уровне метамоделей. В основе ATL лежит стандартизованный язык описания ограничений OCL, что делает возможным использование в процессе описания трансформаций все преимущества языка OCL: стандартные типы, фильтры, помощники и др.

К ограничениям данного языка следует отнести высокие требования, предъявляемые к уровню квалификации разработчика преобразования. Поскольку ATL в большинстве случаев использует лишь текстовое определение трансформации, то, помимо знания исходной и целевой метамоделей, разработчик должен знать сам язык описания преобразований. Кроме того, использование императивных конструкций элиминирует преимущества декларативного подхода. Отсутствие навигации по целевой модели затрудняет процесс определения правил преобразования.

ATL является диалектом языка QVT и, как следствие, наследует все его ограничения. Одним из отличий от QVT является очень жесткое ограничение на создаваемые трансформации: в левой части правила в ATL допускается наличие лишь одного элемента. Это сильно усложняет описание правил, увеличивая их количество. Все это делает невозможным использование данного подхода в системе MetaLanguage.

Однако ATL имеет и существенное преимущество по сравнению с другими подходами. В рамках ATL достаточно просто реализуется трансформация ограничений, описанных на OCL. Такие ограничения могут транслироваться в новую модель, оставаясь корректными для неё.

Основная цель MTBE – автоматическая генерация правил трансформации на основе начального набора образцов. Однако ни одна из реализаций данного подхода не гарантирует, что генерация правил

преобразования моделей корректна и полна. Более того, сгенерированные правила преобразования сильно зависят от исходного набора образцов.

Текущие реализации подхода MTBE позволяет выполнять лишь полные эквивалентные отображения атрибутов, не учитывая их сложные преобразования.

Заключение

Подводя итог, можно говорить о том, что все рассмотренные подходы обладают теми или иными недостатками, которые ограничивают их применимость для описания трансформаций в системе MetaLanguage. Наиболее подходящим и перспективным, с точки зрения авторов, является алгебраический подход с одинарным выталкиванием при условии внесения в него следующих расширений:

- наличие возможности многоуровневого описания метамоделей из левой и правой части правила;
- описание правил трансформаций должно производиться на одном уровне иерархии, а их применение – на другом;
- наличие возможности преобразования модели, созданной с использованием одного языка моделирования, в эквивалентную модель, описанную на другом языке;
- наличие возможности описания трансформаций вида «модель-модель» и «модель-текст»;
- наличие возможности трансформации атрибутов элементов метамодели.

Но не следует забывать о преимуществах рассмотренных подходов, некоторые из них способны сделать компонент трансформации моделей системы MetaLanguage достаточно мощным инструментарием.

Библиографический список

1. *Кузнецов М.Б.* Трансформация UML-моделей и ее использование в технологии MDA // Программирование. 2007. Вып. 33. С. 65-79.
2. *Миков А.И., Борисов А.Н.* Графовые грамматики в автономном мобильном компьютинге // Математика программных систем: межвуз. сб. науч. ст. Пермь: Изд-во Перм. гос. нац. исслед. ун-та, 2012. Вып. 9. С. 50-59.

3. *Поляков В.А., Брыксин Т.А.* Разработка визуального интерпретатора моделей в системе QReal // Материалы межвуз. конкурса-конференции студентов, аспирантов и молодых ученых Северо-Запада «Технологии Miscrosoft в теории и практике программирования». СПб.: Изд-во СПбГПУ, 2011. С. 58.
4. *Ступников С.А., Калиниченко Л.А.* Методы автоматизированного построения трансформаций информационных моделей // Системы и средства информатики. 1. 2009. № 19. С. 34-62.
5. *Сухов А.О., Серый А.П.* Использование графовых грамматик для трансформации моделей // Материалы конференции «CSEDays 2012». Екатеринбург: Изд-во Урал. ун-та, 2012. С. 48-55.
6. *Agrawal A., Karsai G., Neema S., Shi F., Vizhanyo A.* The design of a language for model transformations // Journal on Software and Systems Modeling. 2006. Vol. 5. P. 261-288.
7. *Agrawal A., Karsai G., Shi F.* Graph Transformations on Domain-Specific Models // International Journal on Software and Systems Modeling. Nashville: Vanderbilt University Press, 2003. P. 1-43.
8. *Balasubramanian D., Narayanan A., Buskirk C.P., Karsai G.* The Graph Rewriting and Transformation Language: GReAT // Electronic Communications of the EASST. 2006. Vol. 1. P. 1-8.
9. *Balogh A., Varro D.* Advanced model transformation language constructs in the VIATRA2 framework // ACM Symposium on Applied Computing. New York: ACM New York, 2006. P. 1280-1287.
10. *Bezivin J., Jouault F., Touzet D.* An Introduction to the ATLAS Model Management Architecture // Research Report № 05.01. LINA, 2005. 24 p.
11. *Chiprianov V., Kermarrec Y., Alff P.D.* An Approach for Constructing a Domain Definition Metamodel with ATL // 1st International Workshop on Model Transformation with ATL. Nantes, 2009. P. 18-33.
12. *Corradini A., Montanari U., Rossi F., Ehrig H., Heckel R., Loewe M.* Algebraic approaches to graph transformation. Part I: Basic concepts and double pushout approach // Handbook of Graph Grammars and Computing by Graph transformation. 1997. Vol. 1. P. 163-246.
13. *Csertan G., Huszerl G., Majzik I., Pap Z., Pataricza A., Varro D.* VIATRA – Visual Automated Transformations for Formal Verifica-

tion and Validation of UML Models // Proceedings of the 17th IEEE international conference on Automated software engineering. Washington: IEEE Computer Society, 2002. P. 267-270.

14. *Edwards J.* Example centric programming // ACM SIGPLAN Notices. 2004. Vol. 39, No. 12. P. 84-91.
15. *Ehrig H., Ehrig K., Prange U., Taentzer G.* Fundamentals of Algebraic Graph Transformation. New York: Springer-Verlag, 2006. 388 p.
16. *Ehrig H., Heckel R., Korff M., Loewe M., Ribeiro L., Wagner A., Corradini A.* Algebraic approaches to graph transformation. Part II: single pushout approach and comparison with double pushout approach // Handbook of Graph Grammars and Computing by Graph transformation. 1997. Vol. 1. P. 247-312.
17. *Gardner T., Griffin C., Koehler J., Hauser R.* A review of OMG MOF 2.0 Query/Views/Transformations Submissions and Recommendations towards the final Standard // Proceedings of the 1st International Workshop on Metamodeling for MDA. York, 2003. P. 1-20.
18. *Grabska E., Strug B.* Applying Cooperating Distributed Graph Grammars in Computer Aided Design // Parallel Processing and Applied Mathematics. 2006. Vol. 3911/2006. P. 567-574.
19. Handbook on Graph Grammars and Computing by Graph Transformation / ed. by G. Rozenberg. New Jersey: World Scientific Publishing Co., 1999. 677 p.
20. *Jouault F., Allilaire F., Bezivin J., Kurtev I.* ATL: A model transformation tool // Science of Computer Programming. 2008. Vol. 72. P. 31-39.
21. *Jouault F., Kurtev I.* Transforming Models with ATL // Proceedings of the 2005 international conference on Satellite Events at the MoDELS 2005 Conference. Berlin: Springer-Verlag, 2006. P. 128-138.
22. *Kessentini M., Sahraoui H., Boukadoum M.* Model Transformation as an Optimization Problem // Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems. Berlin: Springer-Verlag, 2008. Vol. 5301/2008. P. 159-173.
23. *Lambers L.* A New Version of GTXL: An Exchange Format for Graph Transformation Systems // Electronic Notes in Theoretical Computer Science. 2004. Vol. 127(1). P. 51-63.

24. *Markovic S., Baar T.* Semantics of OCL specified with QVT // Software and Systems Modeling. 2008. Vol. 7, No. 4. P. 399-422.

25. *Mens T., Czarnecki K., Gorp P.V.* A Taxonomy of Model Transformations // Electronic Notes in Theoretical Computer Science. Amsterdam: Elsevier Science Publishers, 2006. Vol. 152. P. 125-142.

26. *Muggleton S., Raedt L.* Inductive logic programming: Theory and methods // Journal Logic Program. 1994. Vol. 19-20. P. 629-679.

27. *Nickel U., Niere J., Ziindorf A.* Tool Demonstration: The FUJABA Environment // Proceedings of the the 22nd International Conference on Software Engineering (ICSE). New York: ACM Press, 2000. P. 1-4.

28. *Sedighizadeh D., Masehian E.* Particle Swarm Optimization Methods, Taxonomy and Applications // International Journal of Computer Theory and Engineering. 2009. Vol. 1, № 5. P. 486-502.

29. *Stevens P.* Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions // Model Driven Engineering Languages and Systems. International Book Series «Lecture Notes in Computer Science». 2007. Vol. 4735/2007. P. 1-15.

30. *Sukhov A.O., Lyadova L.N.* MetaLanguage: a Tool for Creating Visual Domain-Specific Modeling Languages // Proc. of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering. Moscow: ISP RAS, 2012. P. 42-53.

31. *Sun Y., White J., Gray J.* Model Transformation by Demonstration / Model Driven Engineering Languages and Systems. Berlin: Springer-Verlag, 2009. P. 712-726.

32. *Taentzer G.* AGG: A Graph Transformation Environment for Modeling and Validation of Software // Applications of Graph Transformations with Industrial Relevance. International Book Series «Lecture Notes in Computer Science». 2004. Vol. 3062/2004. P. 446-453.

33. *Tratt L.* Model transformations and tool integration // Journal on Software and System Modeling. 2005. Vol. 4. P. 112-122.

34. *Varro D.* Automated Model Transformations for the Analysis of IT Systems: Ph.D. thesis. Budapest: Budapest University of Technology and Economics, 2003. 240 p.

35. *Varro D., Balogh Z.* Automating Model Transformation by Example Using Inductive Logic Programming // Proceedings of the 2007

ACM Symposium on Applied Computing. New York: ACM Press, 2007. P. 978-984.

36. *Varro D., Pataricza A.* VPM: A Visual, Precise and Multilevel Metamodeling Framework for Describing Mathematical Domains and UML // Journal of Software and Systems Modeling. 2003. Vol. 2(3). P. 187-210.
37. *Vizhanyo A.* Improving the Usability of a Graph Transformation Language // Electronic Notes in Theoretical Computer Science. 2006. Vol. 152. P. 207-222.
38. *Wimmer M., Strommer M., Kargl H., Kramler G.* Towards Model Transformation Generation By-Example // Proceedings of the 40th Annual Hawaii International Conference on System Sciences. Washington: IEEE Computer Society, 2007. P. 1-10.
39. *Zloof M.M.* Query-by-example: the invocation and definition of tables and forms // VLDB '75 Proceedings of the 1st International Conference on Very Large Data Bases. New York: ACM Press, 1975. P. 1-24.

А.О. Сухов, Л.Н. Лядова

Национальный исследовательский университет
«Высшая школа экономики»
(Пермский филиал)

ASuhov@hse.ru, LLyadova@hse.ru

АЛГОРИТМЫ ФУНКЦИОНИРОВАНИЯ И ОПЕРАЦИИ НАД МОДЕЛЯМИ В СИСТЕМЕ METALANGUAGE[3]

Система MetaLanguage – это языковой инструментарий, предназначенный для создания визуальных динамически настраиваемых предметно-ориентированных языков моделирования [2, 3, 7]. Процесс создания и трансформации моделей предметной области с использованием системы MetaLanguage схематично представлен на рис. 1.

Как видно из рисунка, для создания нового предметно-ориентированного языка следует задать основные конструкции языка: определить сущности и отношения между ними, задать ограничения, налагаемые на сущности и отношения метамодели. При этом конкретный синтаксис языка задается в процессе описания абстрактного синтаксиса.

После построения метамодели (модели языка моделирования) разработчик получает в распоряжение готовый расширяемый настраиваемый визуальный язык моделирования. Используя полученный DSL, пользователь может создавать модели, содержащие конкретные объекты предметной области и связи между ними.

Если в процессе построения модели оказалось, что в описании языка были допущены неточности, то пользователь может на любом этапе вернуться к его описанию и внести необходимые изменения без повторной генерации кода и перезапуска системы, при этом система автоматически выполнит все необходимые каскадные изменения в соответствующих моделях.

[3] Работа выполнена при поддержке РФФИ (проект № 12-07-00763-а)

После построения модели необходимо проверить, удовлетворяет ли она ограничениям метамодели, с помощью которой описана эта модель. Для этого следует выполнить валидацию созданной модели. Если какие-либо из ограничений не выполняются, то пользователь будет об этом проинформирован. Если ошибок в созданной модели не обнаружено, пользователь может приступить к описанию трансформаций модели, т.е. выполнить ее преобразование в эквивалентную модель, описанную в иной текстовой/графической нотации [1, 6].

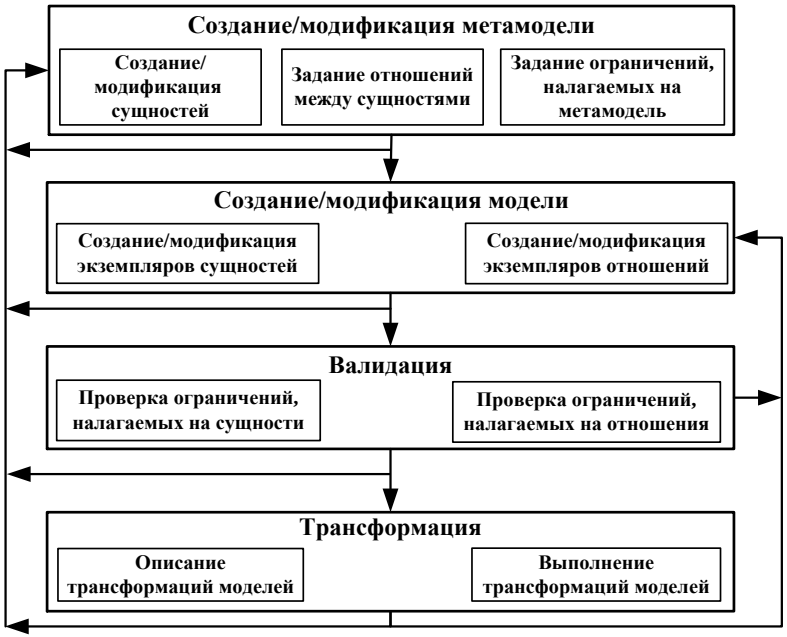


Рис. 1. Процесс создания/модификации моделей предметной области с использованием системы MetaLanguage

Рассмотрев процесс создания моделей предметной области с использованием системы MetaLanguage, перейдем к описанию алгоритмов, реализующих основные этапы этого процесса.

Для формального описания процесса создания визуальных предметно-ориентированных языков была построена математическая модель, в которой даны формальные определения метамодели и модели, описаны операции создания графа модели и его интерпретации [4, 5].

Построение графа метамодели

Создание метамодели – это построение такого графа метамодели $GMM = (V, E)$, для которого в начальный момент времени $V = \varnothing$, $E = \varnothing$, а в процессе добавления сущностей языка моделирования, отношений между ними и ограничений, налагаемых на сущности и отношения, множества V и E пополняются новыми элементами в соответствии с правилами создания сущностей, отношений и ограничений. Проследим, как меняется граф GMM при создании различных конструкций метаязыка системы MetaLanguage.

Создание сущности

Определим операцию создания сущности. Пусть в граф метамодели $GMM' = (V', E')$ необходимо добавить сущность, которой соответствует вершина ent'. Введём следующие обозначения:

- $EAttr'$ – множество вершин графа метамодели, соответствующих атрибутам создаваемой сущности;
- $ERest'$ – множество вершин графа метамодели, соответствующих ограничениям, налагаемым на создаваемую сущность;
- eea' – дуга графа метамодели, соединяющая вершину ent' с множеством вершин $EAttr'$;
- eer' – дуга графа метамодели, соединяющая вершину ent' с множеством вершин $ERest'$.

Создание сущности – это операция над графом метамодели GMM', результатом выполнения которой является такой граф метамодели $GMM = (V, E)$, что

$$V = V' \bigcup \{ent'\} \bigcup EAttr' \bigcup ERest',\ E = E' \bigcup \{eea'\} \bigcup \{eer'\}.$$

В соответствии с алгоритмом 1 при создании сущности в множество вершин графа метамодели необходимо включить вершину ent', определяющую создаваемую сущность, множество вершин, соответствующих атрибутам, ограничениям создаваемой сущности. Помимо этих вершин, необходимо создать дуги, соединяющие вершину ent' с каждым из множеств $EAttr'$, $ERest'$.

При создании сущности в граф метамодели будут добавлены $|EAttr'| + |ERest'| + 1$ вершина и две дуги, т.е. сложность алгоритма добавления новой сущности в метамодель линейна и равна $O(|EAttr'| + |ERest'|)$.

Алгоритм 1. Добавление новой сущности в метамодель

```
Добавить_новую_вершину(ent′,V′);
foreach (eattr′_i ∈ EAttr′)
    Добавить_новую_вершину(eattr′_i,V′);
Добавить_новую_дугу(eea′,E′);
foreach (erest′_i ∈ ERest′)
    Добавить_новую_вершину(erest′_i,V′);
Добавить_новую_дугу(eer′,E′);
```

Создание отношения

После описания сущностей языка моделирования, необходимо задать отношения между ними. Поскольку метамодель может содержать отношения трех типов (наследование, ассоциация и агрегация), рассмотрим создание каждого из типов отношений отдельно.

Создание отношения наследования. Смысл отношения данного типа заключается в наследовании атрибутов, ограничений, отношений сущности-родителя. Наследование атрибутов и ограничений реализовано посредством их копирования. Это обосновано тем, что вершины, соответствующие этим характеристикам сущности, сгруппированы, поэтому при добавлении собственных атрибутов и ограничений дочерней сущности не требуется создание нового множества вершин и дополнительной дуги. Наследование отношений, в которых участвует родительская сущность, реализуется созданием дуг графа метамодели от вершин, соответствующих дочерним сущностям, к вершинам, соответствующим наследуемым отношениям. Таким образом, при создании отношения наследования в множество атрибутов, ограничений сущности-потомка необходимо добавить вершины, соответствующие атрибутам, ограничениям сущности-родителя, а также провести дуги к вершинам, которые соответствуют отношениям, наследуемым у сущности-родителя. Поскольку при построении модели экземпляр отношения наследования создать нельзя, то собственных атрибутов и ограничений данный тип отношения не имеет.

Пусть в граф метамодели $GMM' = (V', E')$ необходимо добавить отношение наследования, которому соответствует вершина rel'. *Отношению наследования* поставим в соответствие вершину графа и две дуги. Первая дуга направлена от вершины, соответствующей дочерней сущности, к вершине графа, соответствующей отношению наследования. Вторая дуга направлена от вершины, соответствующей

отношению наследования, к вершине, соответствующей родительской сущности.

Введём следующие обозначения:

- ent_i – вершина графа метамодели, соответствующая родительской сущности;
- ent_{i-1} – вершина графа метамодели, соответствующая дочерней сущности;
- $EAttr_i'$ – множество вершин графа метамодели, соответствующих атрибутам родительской сущности;
- $ERest_i'$ – множество вершин графа метамодели, соответствующих ограничениям, налагаемым на родительскую сущность;
- $EAttr_{i-1}'$ – множество вершин графа метамодели, соответствующих атрибутам дочерней сущности;
- $ERest_{i-1}'$ – множество вершин графа метамодели, соответствующих ограничениям дочерней сущности;
- eea_{i-1}' – дуга метамодели, соединяющая вершину ent_{i-1} с множеством вершин, соответствующих атрибутам дочерней сущности;
- eer_{i-1}' – дуга графа метамодели, соединяющая вершину ent_{i-1} с множеством вершин, соответствующих ограничениям, налагаемым на дочернюю сущность;
- Rel_i' – множество дуг графа метамодели, соединяющих вершину ent_{i-1} с вершинами графа метамодели, соответствующими отношениям, в которых участвует родительская сущность.

Создание отношения наследования – это операция над графом метамодели $GMM' = (V', E')$, результатом выполнения которой является такой граф метамодели $GMM = (V, E)$, что

$$V = V' \cup \{rel'\} \cup RAttr_i' \cup RRest_i',$$

$$E = E' \cup \{eea_{i-1}'\} \cup \{eer_{i-1}'\} \cup Rel_i' \cup \{(ent_{i-1}, rel')\} \cup \{(rel', ent_i)\}.$$

При создании отношения наследования в метамодель будут добавлены $|RAttr_i'| + |RRest_i'| + 1$ вершина и $|Rel_i'| + 4$ дуги. Сложность алгоритма создания отношения наследования (см. алгоритм 2) равна $O(|RAttr_i'| + |RRest_i'| + |Rel_i'|)$.

Алгоритм 2. Создание отношения наследования

```
Добавить_новую_вершину(rel′,V′);
foreach (eattr′_ij ∈ EAttr′_i)
   Добавить_новую_вершину(eattr′_ij,EAttr′_i-1);
foreach (erest′_ij ∈ ERest′_i)
   Добавить_новую_вершину(erest′_ij,ERest′_i-1);
if (eea′_i-1 ∉ E′)
   Добавить_новую_дугу(eea′_i-1,E′);
if (eer′_i-1 ∉ E′)
   Добавить_новую_дугу(eer′_i-1,E′);
foreach (rel′_ij ∈ Rel′_i)
   Добавить_новую_дугу(rel′_ij,E′);
Добавить_новую_дугу((ent_i-1,rel′),E′);
Добавить_новую_дугу((rel′,ent_i),E′);
```

Рассмотрим пример. Опишем фрагмент графа *GMM*, соответствующий метамодели ERD-диаграмм, изображенной на рис. 2.

Метамодель содержит абстрактную сущность «Абстрактный класс», которая имеет атрибуты «Имя», «Описание». Данная сущность выступает в качестве родительской для сущности «Сущность», которая не имеет собственных атрибутов (см. рис. 3).

Теперь в метамодели создадим отношение наследования между сущностями «Абстрактный класс» и «Сущность» (см. рис. 4).

Как видно из рис. 4, помимо дуг пути «Сущность» – «Наследование» – «Абстрактный класс» (все дуги пути имеют метку «*inh*»), в графе метамодели было произведено копирование наследуемых атрибутов $EAttr_{i-1} = \{$«*Имя*», «*Описание*»$\}$ и проведена дуга от сущности «Сущность» к множеству этих атрибутов.

Создание отношения ассоциации. Пусть в граф метамодели $GMM' = (V', E')$ необходимо добавить отношение ассоциации, которому соответствует вершина rel'. Пусть ent_{i-1}, ent_{i+1} – вершины, соответствующие соединяемым сущностям.

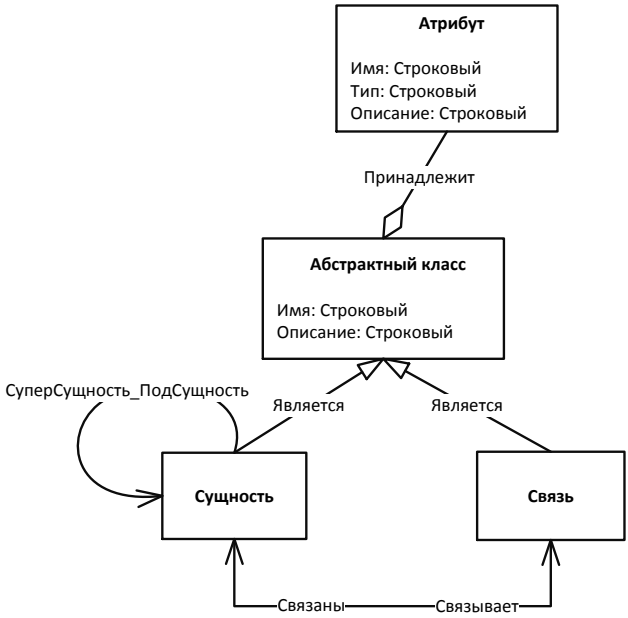


Рис. 2. Фрагмент метамодели диаграмм «Сущность-Связь»

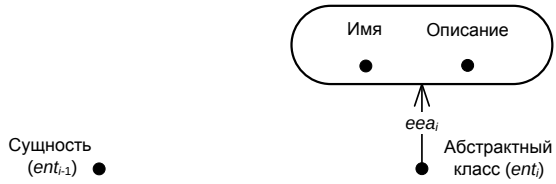


Рис. 3. Создание отношения наследования в графе метамодели. Этап I

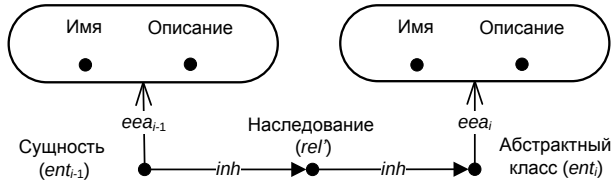


Рис. 4. Создание отношения наследования в графе метамодели. Этап II

Поскольку отношение «Ассоциация» может являться как однонаправленным, так и двунаправленным, то при создании этого отношения в граф метамодели для однонаправленной ассоциации будут добавлены дуги лишь прямого пути $(ent_{i+1}, rel', ent_{i-1})$, а для двунаправленной ассоциации, помимо дуг прямого пути $(ent_{i+1}, rel', ent_{i-1})$, будут добавлены дуги обратного пути $(ent_{i-1}, rel', ent_{i+1})$.

Кроме того, в граф метамодели необходимо добавить атрибуты и ограничения, налагаемые на создаваемое отношение, а также дуги, соединяющие вершину, соответствующую отношению ассоциации, с множествами вершин, соответствующих атрибутам и ограничениям.

Введём следующие обозначения:

- $RAttr'$ – множество вершин графа метамодели, соответствующих атрибутам создаваемого отношения;
- $RRest'$ – множество вершин графа метамодели, соответствующих ограничениям, налагаемым на создаваемое отношение ассоциации;
- era' – дуга графа метамодели, соединяющая создаваемое отношение с множеством вершин, соответствующих его атрибутам;
- err' – дуга графа метамодели, соединяющая создаваемое отношение с множеством вершин, соответствующих ограничениям, налагаемым на него.

Создание отношения ассоциации – это операция над графом метамодели GMM', результатом выполнения которой является такой граф метамодели $GMM = (V, E)$, что $V = V' \cup \{rel'\} \cup RAttr' \cup RRest'$, а множество дуг определяется следующим образом:

- $E = E' \cup \{era'\} \cup \{err'\} \cup \{(ent_{i+1}, rel')\} \cup \{(rel', ent_{i-1})\}$, если ассоциация однонаправлена;
- $E = E' \cup \{era'\} \cup \{err'\} \cup \{(ent_{i+1}, rel')\} \cup \{(rel', ent_{i-1})\} \cup$ $\cup \{(ent_{i-1}, rel')\} \cup \{(rel', ent_{i+1})\}$, если ассоциация двунаправлена.

При создании отношения ассоциации в метамодель будут добавлены $|RAttr'| + |RRest'| + 1$ вершина, четыре дуги в случае однонаправленной ассоциации и шесть дуг в случае двунаправленной (см. алгоритм 3). Сложность алгоритма создания отношения ассоциации равна $O(|RAttr'| + |RRest'|)$.

Алгоритм 3. Создание двунаправленного отношения ассоциации

```
Добавить_новую_вершину(rel′,V′);
foreach (rattr′_i ∈ RAttr′)
   Добавить_новую_вершину(rattr′_i,V′);
Добавить_новую_дугу(era′,E′);
foreach (rrest′_i ∈ RRest′)
   Добавить_новую_вершину(rrest′_i,V′);
Добавить_новую_дугу(err′,E′);
Добавить_новую_дугу((ent_{i+1},rel′),E′);
Добавить_новую_дугу((rel′,ent_{i-1}),E′);
Добавить_новую_дугу((ent_{i-1},rel′),E′);
Добавить_новую_дугу((rel′,ent_{i+1}),E′);
```

Расширим граф метамодели диаграмм «Сущность-Связь» сущностью «Связь», являющейся дочерней для сущности «Абстрактный класс». Соединим в графе метамодели сущности «Сущность» и «Связь» двунаправленной ассоциацией (см. рис. 5).

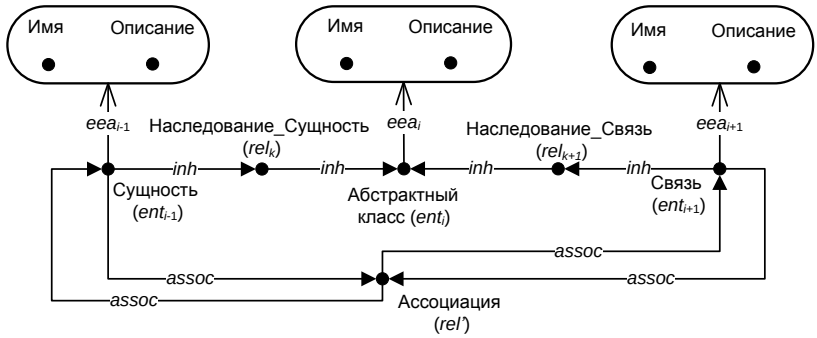


Рис. 5. Создание отношения ассоциации в графе метамодели

Из рисунка видно, что в графе метамодели *GMM* отношению «Ассоциация» соответствует четыре дуги: (ent_{i-1}, rel'), (rel', ent_{i+1}), (ent_{i+1}, rel'), (rel', ent_{i-1}), все они имеют метку «*assoc*».

Создание отношения агрегации. Определим операцию создания отношения агрегации. Пусть в граф метамодели $GMM' = (V', E')$ необходимо добавить отношение агрегации, которому соответствует вершина rel'.

Агрегация, в отличие от ассоциации, моделирует отношение «часть-целое» и изображается при построении метамодели в виде однонаправленной дуги, поэтому при ее создании в граф метамодели необходимо добавить лишь дуги прямого пути, проведенного от вершины, соответствующей сущности «часть», к вершине, соответствующей данному отношению, а далее – к вершине, соответствующей сущности «целое».

Введём следующие обозначения:

- ent_i – вершина графа метамодели, соответствующая сущности «целое»;
- ent_{i+2} – вершин графа метамодели, соответствующая сущности «часть»;
- $RAttr'$ – множество вершин графа метамодели, соответствующих атрибутам создаваемого отношения агрегации;
- $RRest'$ – множество вершин графа метамодели, соответствующих ограничениям, налагаемым на создаваемое отношение;
- era' – дуга графа метамодели, соединяющая создаваемое отношение с множеством вершин, соответствующих его атрибутам;
- err' – дуга графа метамодели, соединяющая создаваемое отношение с множеством вершин, соответствующих ограничениям, налагаемым на него.

Создание отношения агрегации – это операция над графом метамодели GMM', результатом выполнения которой является такой граф метамодели $GMM = (V, E)$, что

$$V = V' \cup \{rel'\} \cup RAttr' \cup RRest',$$

$$E = E' \cup \{era'\} \cup \{err'\} \cup \{(ent_{i+2}, rel')\} \cup \{(rel', ent_i)\}.$$

В соответствии с алгоритмом 4, при создании отношения агрегации в метамодель будут добавлены $|RAttr'| + |RRest'| + 1$ вершина и четыре дуги. Сложность алгоритма добавления в метамодель отношения агрегации равна $O(|RAttr'| + |RRest'|)$.

Алгоритм 4. Создание отношения агрегации

```
Добавить_новую_вершину(rel′,V′);
foreach (rattr′_i ∈ RAttr′)
   Добавить_новую_вершину(rattr′_i,V′);
Добавить_новую_дугу(era′,E′);
foreach (rrest′_i ∈ RRest′)
   Добавить_новую_вершину(rrest′_i,V′);
Добавить_новую_дугу(err′,E′);
Добавить_новую_дугу((ent_{i+2},rel′),E′);
Добавить_новую_дугу((rel′,ent_i),E′);
```

Рассмотрим пример. Пусть в метамодель ERD-диаграмм необходимо добавить сущность «Атрибут» с атрибутами «Имя», «Тип», «Описание». Причем сущность «Атрибут» связана агрегацией с сущностью «Абстрактный класс» (см. рис. 6).

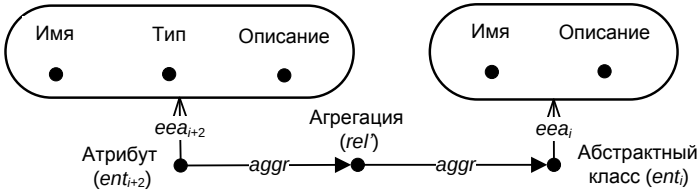


Рис. 6. Создание отношения агрегации в графе метамодели

Как видно из рис. 6, при создании агрегации между сущностями «Атрибут» и «Абстрактный класс» в граф метамодели были добавлены дуги пути «Атрибут» – «Агрегация» – «Абстрактный класс» (на рисунке все дуги пути имеют метку «aggr»).

Создание ограничений

Пусть в графе метамодели $GMM' = (V', E')$ необходимо задать ограничение $rest'$, налагаемое на сущность (отношение) ent'. Обозначим дугу метамодели, соединяющую вершину ent' с множеством вершин, соответствующих ограничениям, налагаемым на ent', через eer'.

Создание ограничения, налагаемого на сущность (отношение) ent' – это операция над графом метамодели GMM', результатом вы-

полнения которой является такой граф метамодели $GMM = (V, E)$, что

$$V = V' \bigcup \{rest'\},\ E = \begin{cases} E' \bigcup \{eer'\}, \text{ если } eer' \notin E'; \\ E', \text{ в противном случае} \end{cases}.$$

Таким образом, при добавлении ограничения, налагаемого на сущность (отношение), в граф метамодели необходимо включить вершину $rest'$, соответствующую данному ограничению, а также дугу, соединяющую сущность (отношение) ent' с множеством вершин, соответствующих ограничениям, на нее (него) налагаемым, если эта дуга ранее не была создана.

Построение графа модели

При *создании модели* необходимо определить основные понятия предметной области, которые являются экземплярами сущностей метамодели, а также связи между понятиями, в качестве которых выступают экземпляры отношений метамодели.

Создание модели – это построение такого графа модели $GM = (VI, EI)$, для которого в начальный момент времени $VI = \varnothing, EI = \varnothing$, а в процессе добавления новых объектов предметной области и связей между ними множества VI, EI пополняются новыми элементами в соответствии с правилами создания экземпляров сущностей и отношений.

Рассмотрим основные операции создания модели и то, как меняется граф модели при выполнении этих операций.

Создание экземпляра сущности

Определим операцию создания экземпляра сущности. Пусть в граф модели $GM' = (VI', EI')$ необходимо добавить экземпляр сущности, которому соответствует вершина $entI'$. Введём следующие обозначения:

- ent' – вершина графа метамодели, соответствующая сущности, экземпляр которой добавляется в модель;
- $EAttrI'$ – множество вершин графа модели, соответствующих значениям атрибутов создаваемого экземпляра сущности;
- $eeaI'$ – дуга графа модели, соединяющая вершину $entI'$ с множеством вершин $EAttrI'$;
- t' – дуга, соединяющая вершину графа модели $entI'$ с верши-

ной графа метамодели ent'.

Дуга t' позволяет поддерживать метамодели и созданные на их основе модели в согласованном состоянии. При внесении изменений в метамодель системы MetaLanguage выполняет все необходимые изменения в соответствующих моделях. Кроме того, наличие данной связи позволяет выполнять интерпретацию моделей, определять типы их элементов, выполнять различные операции над этими моделями.

Создание экземпляра сущности – это операция над графом модели GM', результатом выполнения которой является такой граф модели $GM = (VI, EI)$, что

$$VI = VI' \bigcup \{entI'\} \bigcup EAttrI' ,\ EI = EI' \bigcup \{eeaI'\} \bigcup \{t'\} .$$

В соответствии с алгоритмом 5, при создании экземпляра сущности в множество вершин модели необходимо включить вершину $entI'$, соответствующую создаваемому экземпляру сущности, а также множество вершин, соответствующих значениям его атрибутов. Помимо этого, необходимо создать дугу, соединяющую вершину $entI'$ с множеством вершин $EAttrI'$, и дугу t', соединяющую создаваемый экземпляр сущности с сущностью, на основе которой создаётся данный экземпляр.

Алгоритм 5. Создание экземпляра сущности

```
Добавить_новую_вершину(entI',VI');
foreach (eattrI'_i ∈ EAttrI')
   Добавить_новую_вершину(eattrI'_i,VI');
Добавить_новую_дугу(eeaI',EI');
Добавить_новую_дугу(t',EI');
```

При создании экземпляра сущности в граф модели будут добавлены $|EAttrI'|+1$ вершина и две дуги, т.е. сложность алгоритма добавления нового экземпляра сущности в модель линейна и равна $O(|EAttrI'|)$.

Создание экземпляра отношения

В соответствии с предложенным подходом в моделях могут быть созданы только экземпляры отношений ассоциации и агрегации. Поскольку отношение агрегации – это частный случай отношения ассоциации, а экземпляр отношения ассоциации всегда является однонаправленным, то достаточно рассмотреть алгоритм создания экземпляра

отношения только одного типа. Проследим, как изменится граф модели при создании экземпляра отношения ассоциации (агрегации).

Определим операцию создания экземпляра отношения ассоциации (агрегации). Пусть в граф модели $GM' = (VI', EI')$ необходимо добавить экземпляр отношения ассоциации (агрегации), которому соответствует вершина $relI'$. Пусть $entI'_i$, $entI'_{i+1}$ – вершины, соответствующие соединяемым экземплярам сущностей, а t' – дуга, соединяющая вершину графа модели $relI'$ с вершиной графа метамодели, соответствующей отношению, экземпляр которого создается. Обозначим через $RAttrI'$ множество вершин графа модели, соответствующих значениям атрибутов создаваемого экземпляра отношения, а через $eraI'$ – дугу графа модели, соединяющую вершину $relI'$ с множеством вершин $RAttrI'$.

Создание экземпляра отношения ассоциации (*агрегации*) – это операция над графом модели GM', результатом выполнения которой является такой граф модели $GM = (VI, EI)$, что

$$VI = VI' \cup \{relI'\} \cup RAttrI',$$

$$EI = EI' \cup \{eraI'\} \cup \{(entI'_{i+1}, relI')\} \cup \{(relI', entI'_i)\} \cup \{t'\}.$$

Таким образом, при создании экземпляра отношения ассоциации (агрегации) в граф модели необходимо добавить вершину, соответствующую создаваемому экземпляру отношения, множество вершин-значений атрибутов экземпляра отношения, дугу $eraI'$, дугу t', соединяющую вершину графа модели, соответствующую создаваемому экземпляру отношения, и вершину графа метамодели, соответствующую отношению, на основе которого создается этот экземпляр, а также дуги пути $(entI'_i, relI', entI'_{i+1})$.

При создании экземпляра отношения ассоциации (агрегации) в граф модели будут добавлены $|RAttrI'|+1$ вершина и четыре дуги (см. алгоритм 6). Сложность алгоритма создания экземпляра отношения ассоциации (агрегации) равна $O(|RAttrI'|)$.

Рассмотрим пример. На рис. 7 изображен граф, представляющий собой объединение графа метамодели ERD-диаграмм и графа модели, созданной с помощью этой метамодели. Построение такого графа возможно в силу того, что для описания графов моделей различных уровней иерархии используется единый формализм – ориентированные псевдо-метаграфы. Для наглядности на рисунке не изображены вершины, соответствующие атрибутам сущностей.

Алгоритм 6. Создание экземпляра отношения ассоциации (агрегации)

```
Добавить_новую_вершину(relI′,VI′);
foreach (attrI′_i ∈ RAttrI′)
   Добавить_новую_вершину(attrI′_i,VI′);
Добавить_новую_дугу(eraI′,EI′);
Добавить_новую_дугу((entI′_{i+1},relI′),EI′);
Добавить_новую_дугу((relI′,entI′_i),EI′);
Добавить_новую_дугу(t′,EI′);
```

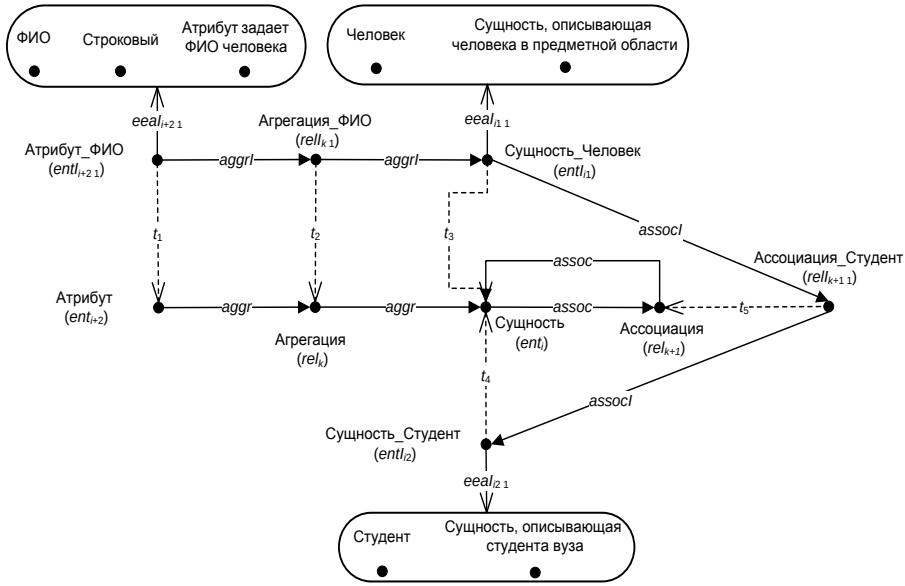


Рис. 7. Граф модели, построенный на основе графа метамодели ERD-диаграмм

На рис. 7 представлены следующие вершины, соответствующие экземплярам сущностей:

- $entI_{i_1}$ – вершина графа модели «Сущность_Человек», соответствующая экземпляру сущности «Сущность»;
- $entI_{i_2}$ – вершина графа модели «Сущность_Студент», соответ-

ствующая экземпляру сущности «Сущность», которая является подсущностью для суперсущности «Сущность_Человек»;

– $entI_{i+2_1}$ – вершина графа модели «Атрибут_ФИО», соответствующая экземпляру сущности «Атрибут».

Каждая из этих вершин соединена дугой с вершинами, соответствующими значениям ее атрибутов.

Модель содержит экземпляры двух отношений, в графе GM им соответствуют вершины:

– $relI_{k_1}$ – экземпляр отношения «Агрегация»;

– $relI_{k+1_1}$ – экземпляр отношения «Ассоциация».

Путь $(entI_{i_1}, relI_{k+1_1}, entI_{i_2})$ определяет экземпляр отношения ассоциации. Вершина $entI_{i+2_1}$ соединена дугой с вершиной $relI_{k_1}$, которая, в свою очередь, смежна вершине $entI_{i_1}$, что отвечает правилам создания экземпляра отношения агрегации в модели.

Дуги t_1, t_2, t_3, t_4, t_5 соединяют экземпляры сущностей и отношений с сущностями и отношениями, на основе которых они созданы. Эти дуги представлены на рисунке пунктирными линиями.

Удаление элементов модели и метамодели

Еще одной операцией над моделью и метамоделью является удаление их элементов, причем при исключении сущностей и отношений необходимо каскадно удалить все их экземпляры.

Удаление элементов модели

Поскольку алгоритмы удаления элементов модели используются при выполнении алгоритмов удаления элементов метамодели, то рассмотрим их в первую очередь. Проследим, что происходит с графом модели при удалении экземпляров сущностей и отношений.

Удаление экземпляра отношения. Определим операцию удаления экземпляра отношения. Пусть из графа модели $GM' = (VI', EI')$ необходимо удалить экземпляр отношения, которому соответствует вершина $relI'$. Для этого необходимо удалить:

– вершину графа модели $relI'$, соответствующую удаляемому экземпляру отношения;

– множество вершин графа модели, соответствующих атрибутам удаляемого экземпляра отношения, – $RAttrI'$;

– дугу графа модели, соединяющую вершину $relI'$ с множеством вершин $RAttrI'$, – $eraI'$;
– дугу t', соединяющую удаляемый экземпляр отношения с отношением, на основе которого он был создан;
– множество дуг графа модели, инцидентных вершине $relI'$, – ERI'.

Удаление экземпляра отношения – это операция над графом модели $GM' = (VI', EI')$, результатом выполнения которой является такой граф модели $GM = (VI, EI)$, что

$$VI = VI' \setminus (\{relI'\} \cup RAttrI'),\ EI = EI' \setminus (\{eraI'\} \cup ERI' \cup \{t'\}).$$

В соответствии с алгоритмом 7, при выполнении операции удаления экземпляра отношения из графа модели будут удалены $|RAttrI'|+1$ вершина и $|ERI'|+2$ дуги, т.е. сложность алгоритма линейна и равна $O(|RAttrI'|+|ERI'|)$.

Алгоритм 7. Удаление экземпляра отношения

```
Удалить_дугу(eraI',EI');
foreach (rattrI'_i ∈ RAttrI')
    Удалить_вершину(rattrI'_i,VI');
foreach (erI'_i ∈ ERI')
    Удалить_дугу(erI'_i,EI');
Удалить_дугу(t',EI');
Удалить_вершину(relI',VI');
```

Удаление экземпляра сущности. Определим операцию удаления экземпляра сущности. Пусть из графа модели $GM' = (VI', EI')$ необходимо удалить экземпляр сущности, которому соответствует вершина $entI'$. Для этого необходимо удалить:
– вершину графа модели $entI'$, соответствующую удаляемому экземпляру сущности;
– множество вершин графа модели, соответствующих значениям атрибутов удаляемого экземпляра сущности, – $EAttrI'$;
– множество экземпляров отношений, в которых участвует удаляемый экземпляр сущности, – $RelI'$, удаление экземпляров отношений осуществляется с помощью алгоритма 7;
– дугу графа модели, соединяющую вершину $entI'$ с множе-

ством вершин $EAttrI'$, – $eeaI'$;

– дугу t', соединяющую удаляемый экземпляр сущности с сущностью, на основе которой он был создан.

Удаление экземпляра сущности – это операция над графом модели $GM' = (VI', EI')$, результатом выполнения которой является такой граф модели $GM = (VI, EI)$, для которого после удаления множества экземпляров отношений $RelI'$ выполняется

$$VI = VI' \setminus (\{entI'\} \cup EAttrI'),\ EI = EI' \setminus (\{eeaI'\} \cup \{t'\}).$$

В соответствии с алгоритмом 8, при удалении экземпляра сущности из графа модели будут удалены $|EAttrI'|+1$ вершина, две дуги, а также $|RelI'|$ экземпляров отношений, т.е. сложность алгоритма удаления экземпляра сущности равна $O(|EAttrI'|+N)$, где N – сложность удаления $|RelI'|$ экземпляров отношений.

Алгоритм 8. Удаление экземпляра сущности

```
Удалить_дугу(eeaI', EI');
foreach (eattrI'_i ∈ EAttrI')
   Удалить_вершину(eattrI'_i, VI');
foreach (relI'_i ∈ RelI')
   Удалить_экземпляр_отношения(relI'_i);
Удалить_дугу(t', EI');
Удалить_вершину(entI', VI');
```

Рассмотрим пример. Удалим из графа, представленного на рис. 7, экземпляр «Сущность_Студент» сущности «Сущность».

Проследим, как изменялся граф модели при удалении экземпляра «Сущность_Студент». На первом этапе из графа модели был удален экземпляр отношения, в котором участвует удаляемый экземпляр сущности, – «Ассоциация_Студент» (см. рис. 8). При его удалении из графа модели были исключены:

– дуги пути $(entI_{i_1}, relI_{k+1_1}, entI_{i_2})$;

– дуга t_5;

– вершина $relI_{k+1_1}$.

На втором этапе из графа модели были удалены:

– дуга $eeaI_{i_{2_1}}$, соединяющая вершину $entI_{i_2}$ с множеством атри-

бутов удаляемого экземпляра сущности;

- множество вершин $EAttrI_{i_2}$, соответствующих значениям атрибутов экземпляра «Сущность_Студент»;
- дуга t_4;
- вершина $entI_{i_2}$, соответствующая удаляемому экземпляру сущности.

Результатом выполнения операции является граф, представленный на рис. 9.

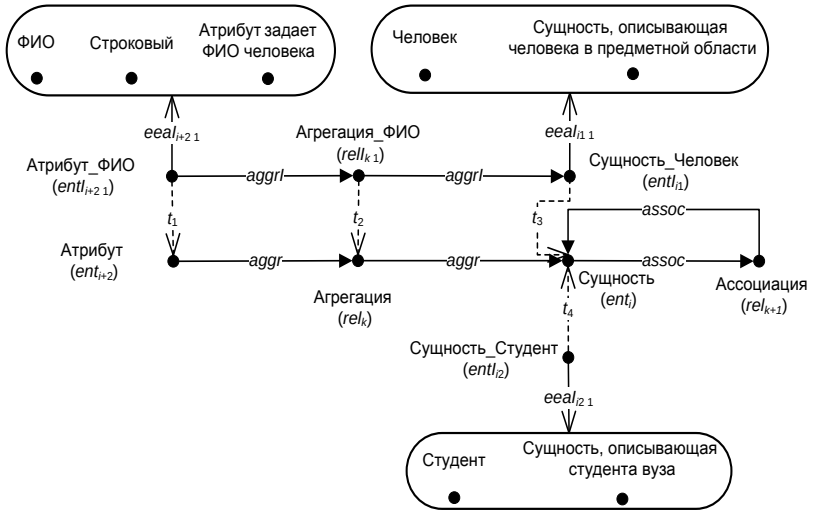


Рис. 8. Удаление экземпляра сущности «Сущность_Студент». Этап I

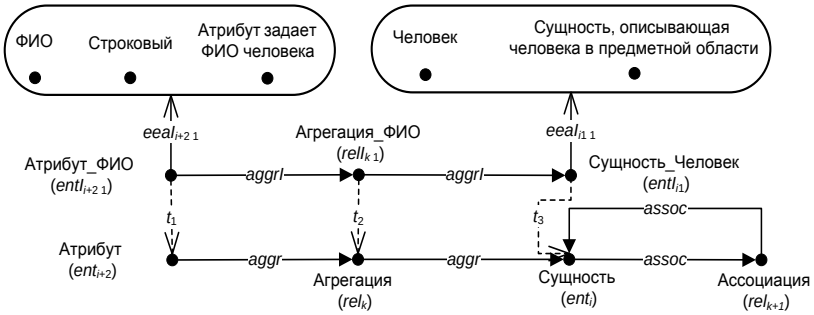


Рис. 9. Удаление экземпляра сущности «Сущность_Студент». Этап II

Удаление элементов метамодели

При рассмотрении алгоритмов удаления элементов метамодели (сущностей, отношений, ограничений) воспользуемся описанными ранее алгоритмами удаления элементов модели. Метамодель может содержать три типа отношений: наследования, ассоциации, агрегации. Поскольку агрегация – это частный случай отношения ассоциации, то семантики этих отношений схожи, отношение наследования имеет семантику, отличную от других типов отношений. Рассмотрим операцию удаления отношений метамодели отдельно для отношений ассоциации (агрегации) и отношений наследования.

Удаление отношений «Ассоциация» и «Агрегация». Определим операцию удаления отношения ассоциации (агрегации). Пусть из графа метамодели $GM' = (VI', EI')$ необходимо удалить отношение ассоциации (агрегации), которому соответствует вершина rel'. Обозначим через k количество моделей, содержащих экземпляры удаляемого отношения.

Для выполнения операции удаления отношения «Ассоциация» («Агрегация») из графа метамодели и графов моделей необходимо удалить:

- вершину графа метамодели, соответствующую удаляемому отношению;
- множество вершин графа метамодели, соответствующих атрибутам удаляемого отношения, – $RAttr'$;
- множество вершин графа метамодели, соответствующих ограничениям, налагаемым на удаляемое отношение, – $RRest'$;
- дугу графа метамодели, соединяющую вершину rel' с множеством вершин $RAttr'$, – era';
- дугу графа метамодели, соединяющую вершину rel' с множеством вершин $RRest'$, – err';
- множество экземпляров удаляемого отношения из всех моделей, созданных на основе изменяемой метамодели, – $\{RelI'_i\}, i = \overline{1,k}$; удаление экземпляров отношения осуществляется с помощью алгоритма 7;
- множество дуг графа метамодели, инцидентных вершине rel', – ER'.

Удаление отношения ассоциации (*агрегации*) – это операция над графом метамодели $GM' = (VI', EI')$ и множеством графов моделей $GM'_i = (VI'_i, EI'_i), i = \overline{1,k}$, результатом выполнения которой является

такой граф метамодели $GMM = (V, E)$, что

$V = V' \setminus (\{rel'\} \cup RAttr' \cup RRest')$, $E = E' \setminus (\{era'\} \cup \{err'\} \cup ER')$,

и графы моделей $GM'_i = (VI'_i, EI'_i), i = \overline{1,k}$, из которых удалены все элементы множеств $\{RelI'_i\}, i = \overline{1,k}$, т.е. все экземпляры удаляемого отношения.

В соответствии с алгоритмом 9, при удалении отношения ассоциации (агрегации) из графа метамодели будут удалены $|RAttr'| + |RRest'| + 1$ вершина и $|ER'| + 2$ дуги, а из графа модели – $\sum_{i=1}^{k} |RelI'_i|$ экземпляров отношения, т.е. сложность данного алгоритма равна $O\left(|RAttr'| + |RRest'| + |ER'| + \sum_{i=1}^{k} |RelI'_i|\right)$.

Алгоритм 9. Удаление отношения ассоциации (агрегации) из метамодели

```
foreach (RelI'_i ⊆ {RelI'_i}, i = 1,k)
foreach (relI'_ij ∈ RelI'_i)
   Удалить_экземпляр_отношения(relI'_ij, GM'_i);
Удалить_дугу(era', E');
foreach (rattr'_i ∈ RAttr')
   Удалить_вершину(rattr'_i, V');
Удалить_дугу(err', E');
foreach (rrest'_i ∈ RRest')
   Удалить_вершину(rrest'_i, V');
foreach (er'_i ∈ ER')
   Удалить_дугу(er'_i, E');
Удалить_вершину(rel', V');
```

Удаление отношения «Наследование». Определим операцию удаления отношения наследования. Пусть из графа метамодели $GM' = (VI', EI')$ необходимо удалить отношение наследования, которому соответствует вершина rel'. Пусть отношение наследования соединяет родительскую сущность, которой соответствует вершина ent_i с дочерней сущностью, которой соответствует вершина ent_{i+1}.

При удалении отношения наследования из графа метамодели и из графов моделей будут удалены:

- вершина графа метамодели, соответствующая удаляемому отношению;
- элементы множества $EAttr'_{i+1}$, которые являются наследованными атрибутами, – $EAttrInh$;
- элементы множества $ERest'_{i+1}$, которые соответствуют ограничениям, наследованным от родительской сущности, – $ERestInh$;
- множество вершин графов моделей, соответствующих значениям атрибутов экземпляров сущности ent_{i+1}, которые она унаследовала от родительской сущности, – $\left\{EAttrI'_j\right\},\ j=\overline{1,k}$;
- дуга графа метамодели eea'_{i+1}, соединяющая вершину ent_{i+1} с множеством $EAttr'_{i+1}$, если оно пусто;
- дуга графа метамодели eer'_{i+1}, соединяющая вершину ent_{i+1} с множеством $ERest'_{i+1}$, если оно пусто;
- множество дуг графа метамодели, инцидентных вершине rel', – ER';
- множество дуг графов моделей $EEAI'$, соединяющих экземпляры дочерней сущности с семейством множеств $\left\{EAttrI'_j\right\},\ j=\overline{1,k}$, если эти множества пусты.

Удаление отношения наследования – это операция над графом метамодели $GM'=(VI',EI')$, множеством графов моделей $GM'_j=(VI'_j,EI'_j),\ j=\overline{1,k}$, результатом выполнения которой является такой граф метамодели $GMM=(V,E)$, что

$$V=V'\setminus\left(\{rel'\}\cup EAttrInh\cup ERestInhI\right),$$

$$E=E'\setminus\left(\{eea'_{i+1}\}\cup\{eer'_{i+1}\}\cup ER'\right),$$

и графы моделей $GM'_j=(VI'_j,EI'_j),\ j=\overline{1,k}$, из которых удалены:

- вершины семейства множеств $\left\{EAttrI'_j\right\},\ j=\overline{1,k}$;
- дуги множества $EEAI'$.

В соответствии с алгоритмом 10, при выполнении операции удаления отношения наследования из графа метамодели будут удалены $\left|EAttrInh\right|+\left|ERestInh\right|+1$ вершина и $\left|ER'\right|+2$ дуги, из графов моделей

– $\sum_{j=1}^{k}\left(\left|EAtrrI'_j\right|\right)$ вершин и $\left|EEAI'\right|$ дуг, т.е. сложность алгоритма удаления отношения наследования равна

$$O\left(\left|EAttrInh\right|+\left|ERestInh\right|+\left|ER'\right|+\sum_{j=1}^{k}\left(\left|EAtrrI'_j\right|\right)+\left|EEAI'\right|\right).$$

Алгоритм 10. Удаление отношения наследования из метамодели

```
foreach ( EAttrI'_j ⊆ {EAttrI'_j}, j = 1,k )
{
   foreach ( eattrI'_j1 ∈ EAttrI'_j )
      Удалить_вершину( eattrI'_j1, VI'_j );
   if ( EAttrI'_j == ∅ )
      Удалить_дугу( eeaI'_j, EI'_j );
}
foreach ( eattr'_ji+1 ∈ EAttrInh )
   Удалить_вершину( eattr'_ji+1, EAttr'_i+1 );
foreach ( erest'_ji+1 ∈ ERestInh )
   Удалить_вершину( erest'_ji+1, ERest'_i+1 );
if ( EAttr'_i+1 == ∅ )
   Удалить_дугу( eea'_i+1, E' );
if ( ERest_i+1' == ∅ )
   Удалить_дугу( eer'_i+1, E' );
foreach ( er'_i ∈ ER' )
   Удалить_дугу( er'_i, E' );
Удалить_вершину( rel', V' );
```

Рассмотрим пример. Удалим из графа метамодели, представленного на рис. 5, отношение «Наследование» между сущностями «Сущность» и «Абстрактный класс» (см. рис. 10).

При выполнении операции из графа метамодели были удалены:

– дуга eea_{i-1_1}, соединяющая вершину, соответствующую дочерней сущности, с множеством ее атрибутов;
– множество атрибутов сущности «Сущность» – $EAttr_{i-1}$;
– дуги пути $(ent_{i-1}, rel_k, ent_i)$;
– вершина rel_k, соответствующая удаляемому отношению.

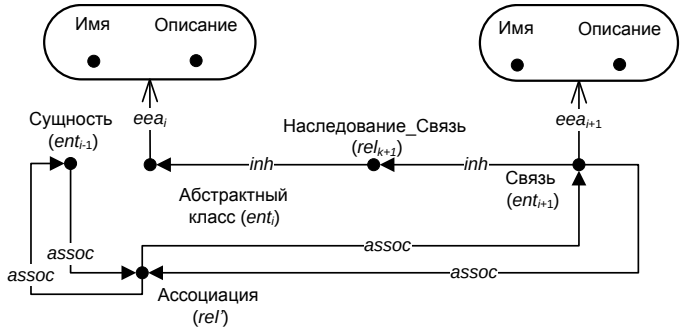


Рис. 10. Удаление отношения «Наследование»

Удаление сущности метамодели. Определим операцию удаления сущности. Пусть из графа метамодели $GMM' = (V', E')$ необходимо удалить сущность, которой соответствует вершина ent'. Обозначим через k количество моделей, содержащих экземпляры удаляемой сущности. Для выполнения операции удаления сущности из графа метамодели и графов моделей необходимо удалить:

- вершину графа метамодели, соответствующую удаляемой сущности;
- множество вершин графа метамодели, соответствующих атрибутам удаляемой сущности, – $EAttr'$;
- множество вершин графа метамодели, соответствующих ограничениям, налагаемым на удаляемую сущность, – $ERest'$;
- множество отношений метамодели, в которых участвует удаляемая сущность, – Rel';
- множество экземпляров удаляемой сущности из всех моделей, созданных на основе изменяемой метамодели, – $\{EntI_i'\}$, $i = \overline{1,k}$, удаление экземпляров сущности осуществляется с помощью алгоритма 8;
- дугу графа метамодели, соединяющую вершину ent' с множеством вершин $EAttr'$, – eea';
- дугу графа метамодели, соединяющую вершину ent' с множеством вершин $ERest'$, – eer'.

Удаление сущности – это операция над графом метамодели $GMM' = (V', E')$ и множеством графов моделей $GM_i' = (VI_i', EI_i'), i = \overline{1,k}$, результатом выполнения которой является

такой граф метамодели $GMM = (V, E)$, для которого после удаления множества отношений Rel' выполняется:

$$V = V' \setminus \left(\{ent'\} \cup EAttr' \cup ERest'\right),\ E = E' \setminus \left(\{eea'\} \cup \{eer'\}\right),$$

и графы моделей $GM'_i = (VI'_i, EI'_i), i = \overline{1,k}$, из которых удалены все элементы семейства множеств $\left\{EntI'_i\right\}, i = \overline{1,k}$, т.е. все экземпляры удаляемой сущности.

В соответствии с алгоритмом 11 при удалении сущности из графа метамодели будут удалены $\left|EAttr'\right| + \left|ERest'\right| + 1$ вершина, две дуги, $\left|Rel'\right|$ отношений, а из графов моделей – $\sum_{i=1}^{k} \left|EntI'_i\right|$ экземпляров сущности. Таким образом, сложность алгоритма удаления сущности равна $O\left(\left|EAttr'\right| + \left|ERest'\right| + M + N\right)$, где M – сложность удаления $\left|Rel'\right|$ отношений, а N – сложность удаления $\sum_{i=1}^{k} \left|EntI'_i\right|$ экземпляров сущностей.

Алгоритм 11. Удаление сущности из метамодели

```
foreach (EntI'_i ⊆ {EntI'_i}, i = 1,k)
foreach (entI'_ij ∈ EntI'_i)
    Удалить_экземпляр_сущности(entI'_ij, GM'_i);
Удалить_дугу(eea', E');
foreach (eattr'_i ∈ EAttr')
    Удалить_вершину(eattr'_i, V');
Удалить_дугу(eer', E');
foreach (rel'_i ∈ Rel')
    Удалить_отношение(rel'_i);
Удалить_вершину(ent', V');
```

Рассмотрим пример. Предположим, что из графа, представленного на рис. 7, необходимо удалить сущность «Сущность», которой соответствует вершина ent_i. На первом этапе из графа модели были удалены все экземпляры сущности «Сущность», в результате был получен граф, изображенный на рис. 11 (в граф добавлены скрытые ранее множества $EAttr_i$, $EAttr_{i+2}$ и дуги eea_i, eea_{i+2}).

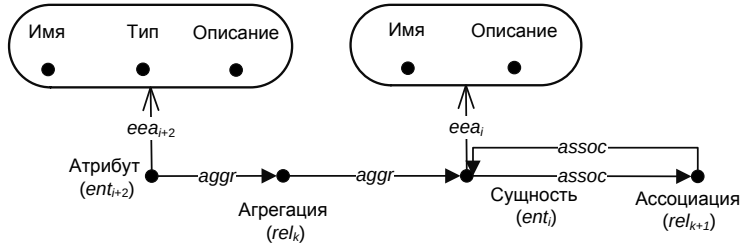


Рис. 11. Удаление сущности «Сущность». Этап I

На втором этапе из метамодели были удалены отношения, в которых участвует сущность «Сущность»: отношения «Агрегация» и «Ассоциация». При этом из графа метамодели были исключены (см. рис. 12):

- дуги путей $(ent_{i+2}, rel_k, ent_i)$, $(ent_i, rel_{k+1}, ent_i)$;
- вершины rel_k, rel_{k+1}.

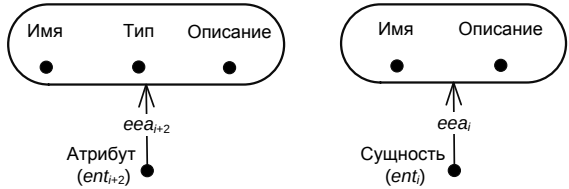


Рис. 12. Удаление сущности «Сущность». Этап II

На третьем этапе из графа метамодели были удалены:

- дуга eea_{i_1}, соединяющую вершину ent_i с множеством вершин $EAttr_i$;
- множество вершин, соответствующих атрибутам удаляемой сущности, – $EAttr_i$;
- вершина ent_i, соответствующая удаляемой сущности.

Результатом выполнения операций является граф, представленный на рис. 13.

Поскольку сущность «Атрибут» связана агрегацией с сущностью «Абстрактный класс», то при удалении сущности «Сущность» она не была удалена.

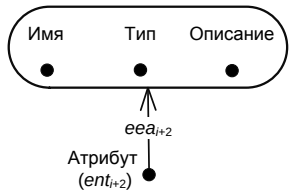


Рис. 13. Удаление сущности «Сущность». Этап III

Удаление ограничений. Пусть в графе метамодели $GMM' = (V', E')$ необходимо удалить ограничение $rest'$, налагаемое на сущность (отношение) ent'. Введём следующие обозначения:

- $ERest'$ – множество вершин, соответствующих ограничениям, налагаемым на ent';
- eer' – дуга, соединяющая вершину ent' с множеством $ERest'$.

Удаление ограничения, налагаемого на сущность (*отношение*) ent' – это операция над графом метамодели $GMM' = (V', E')$, результатом выполнения которой является такой граф метамодели $GMM = (V, E)$, что

$$V = V' \setminus \{rest'\},\ E = \begin{cases} E' \setminus \{eer'\}, \text{ если } ERest' = \{rest'\}; \\ E', \text{ в противном случае.} \end{cases}$$

При удалении ограничения, налагаемого на сущность (отношение), из графа метамодели необходимо удалить вершину $rest'$, соответствующую данному ограничению, а также дугу, соединяющую вершину ent' с множеством ограничений, налагаемых на изменяемую сущность (отношение), если это множество стало пустым.

Проверка ограничений

Проверка ограничений, налагаемых на метамодель, реализуется посредством функции над графом модели $GM = (VI, EI)$, которая возвращает значение ε такое, что

$$\varepsilon = \begin{cases} true, \text{ если присутствует нарушение ограничения;} \\ false, \text{ при отсутствии нарушений.} \end{cases}$$

Некоторые виды ограничений проверяются при создании сущностей и отношений, другие – непосредственно после создания модели. Рассмотрим алгоритмы проверки каждого из видов ограничений более подробно.

Проверка ограничений, налагаемых на сущности

Уникальность имени экземпляра сущности проверяется непосредственно при ее создании. В случае, если задано значение «уникальна в рамках метамодели», то будут просмотрены все экземпляры сущности во всех моделях, построенных на основе данной метамодели (см. алгоритм 12). Если сущность «уникальна в рамках модели», то достаточно просмотреть все экземпляры сущности только текущей модели.

Сложность алгоритма проверки уникальности имени экземпляра сущности равна $O\left(\sum_{k=1}^{|M'|}\sum_{i=1}^{|Ent_k|}\left|EntI_{k_i}\right|\right)$.

Алгоритм 12. Проверка уникальности имени экземпляра сущности

```
switch (entI.EUnique)
{
    case "уникальна в рамках метамодели":
        foreach (m'_k ∈ M')
           foreach (ent_ki ∈ m'_k)
           foreach (entI_ki1 ∈ ent_ki)
              if (entI.EName == entI_ki1.EName)
              {
                    Выдать_сообщение_об_ошибке();
                    ε = true;
              }
              break;
    case "уникальна в рамках модели":
        foreach (ent_ki ∈ m'_k)
           foreach (entI_ki1 ∈ ent_ki)
              if (entI.EName == entI_ki1.EName)
              {
                   Выдать_сообщение_об_ошибке();
                   ε = true;
              }
          break;
}
```

Ограничения на количество экземпляров сущности в модели, как и ограничения на уникальность имени экземпляра, проверяются непосредственно при создании экземпляра сущности. В случае, если число

уже созданных экземпляров сущности определенного типа равно максимально допустимому числу экземпляров, то новый экземпляр создан не будет.

Проверка ограничений на значения атрибутов экземпляров сущностей происходит во время валидации модели. При этом ограничения, налагаемые на каждую сущность метамодели, будут проверены для всех ее экземпляров (см. алгоритм 13).

Алгоритм 13. Проверка ограничений, налагаемых на значения атрибутов экземпляров сущностей

```
foreach (ent_i ∈ Ent)
    foreach (erest_ij ∈ ERest_i)
        foreach (entI_ik ∈ EntI_i)
            if (Ограничение_не_выполняется(erest_ij , entI_ik ))
            {
                Выдать_сообщение_об_ошибке();
                ε = true;
            }
```

Сложность алгоритма проверки ограничений, налагаемых на значения атрибутов экземпляров сущностей, является полиномиальной и равна $O\left(\sum_{i=1}^{|Ent|}\left|ERest_i\right|\cdot\left|EntI_i\right|\right)$.

Проверка ограничений, налагаемых на отношения

Ограничения на уникальность имени экземпляра отношения проверяются непосредственно при его создании. Алгоритм проверки ограничений этого вида аналогичен алгоритму проверки ограничений на уникальность имени экземпляра сущности (см. алгоритм 6) за исключением следующего: вместо обхода по экземплярам сущностей производится обход по экземплярам отношений.

Ограничения на типы соединяемых экземпляров сущностей проверяются при создании экземпляра отношения. Как только пользователь проводит связь между двумя экземплярами сущностей, система определяет их тип и проверяет, существует ли в метамодели отношение между сущностями этого типа. Если такое отношение существует, система позволит создать связь, в противном случае будет выдано сообщение об ошибке.

При проверке ограничений на кратность отношений система для

каждого из соединяемых экземпляров сущностей подсчитывает количество экземпляров сущностей данного типа в модели, уже участвующих в этом отношении, и в случае, если это число будет равно кратности создаваемого экземпляра отношения, то будет выдано сообщение об ошибке, и новый экземпляр отношения создан не будет.

Проверка ограничений, налагаемых на значения атрибутов соединяемых экземпляров сущностей, проводится на этапе валидации в соответствии с алгоритмом 14.

Алгоритм 14. Проверка ограничений, налагаемых на значения атрибутов соединяемых экземпляров сущностей

```
foreach ( rel_ki ∈ m'_k )
   foreach ( relI_kij ∈ rel_ki )
   {
   if (Ограничение_не_выполняется( relI_kij , rel_ki .источник))
       {
         Выдать_сообщение_об_ошибке();
         ε = true;
       }
   if (Ограничение_не_выполняется( relI_kij , rel_ki .приемник))
       {
         Выдать_сообщение_об_ошибке();
         ε = true;
       }
   }
```

Сложность алгоритма проверки ограничений, налагаемых на значения атрибутов соединяемых экземпляров сущностей, равна $O\left(\sum_{i=1}^{|Rel_k|}\left|RelI_{k_i}\right|\right)$.

Библиографический список

1. *Лядова Л.Н., Серый А.П., Сухов А.О.* Подходы к описанию вертикальных и горизонтальных трансформаций метамоделей // Математика программных систем: межвуз. сб. науч. ст. Пермь: Изд-во Перм. гос. нац. исслед. ун-та, 2012. Вып. 9. С. 33-49.
2. *Сухов А.О.* Инструментальные средства создания визуальных предметно-ориентированных языков моделирования // Фундаментальные исследования. 2013. № 4 (ч. 4). С. 848-852.

3. *Сухов А.О.* Среда разработки визуальных предметно-ориентированных языков моделирования // Математика программных систем: межвуз. сб. науч. ст. Пермь: Изд-во Перм. гос. ун-та, 2008. Вып. 5. С. 84-94.
4. *Сухов А.О.* Теоретические основы разработки DSL-инструментария с использованием графовых грамматик // Информатизация и связь. 2011. № 3. С. 35-37.
5. *Сухов А.О.* Формальное описание метаязыка системы MetaLanguage // Труды всерос. научно-практической конференции «Современные проблемы математики и ее прикладные аспекты». Пермь: Изд-во Перм. гос. ун-та, 2010. С. 154-159.
6. *Sukhov A.O., Lyadova L.N.* Horizontal Transformations of Visual Models in MetaLanguage System // Proc. of the 7th Spring/Summer Young Researchers' Colloquium on Software Engineering. Moscow: ISP RAS, 2013. P. 31-40.
7. *Sukhov A.O., Lyadova L.N.* MetaLanguage: a Tool for Creating Visual Domain-Specific Modeling Languages // Proc. of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering. Moscow: ISP RAS, 2012. P. 42-53.

А.О. Сухов, Н.А. Семков

Национальный исследовательский университет
«Высшая школа экономики»

ASuhov@hse.ru, B-boysima@mail.ru

ПРЕДМЕТНО-ОРИЕНТИРОВАННЫЙ ЯЗЫК ОПИСАНИЯ МОДЕЛЕЙ СИСТЕМ ТИПА «УМНЫЙ ДОМ»[4]

С развитием современных цифровых и информационных технологий окружающая нас техника становится все более интеллектуальной. Одним из проектов, позволяющих упростить и автоматизировать повседневную деятельность людей, является система «Умный дом» (Цифровой дом, Smart House, Intelligent Building и др.), представляющая собой сложный комплекс различных взаимодействующих подсистем управления домом. Системы типа «Умный дом» позволяют не только сделать комфортным пребывание человека в доме, но и повысить надежность управления всеми системами жизнеобеспечения, а также эффективность их функционирования, сократив при этом потребление электроэнергии.

Особенностью большинства систем такого типа является объединение отдельных подсистем различных производителей в единый управляемый комплекс, при этом современные технологии позволяют создавать систему покомпонентно, выбирая только тот набор элементов, который необходим клиенту.

Контроль над всеми подсистемами осуществляет центральная панель, запрограммированная владельцем дома в соответствии с его потребностями. Для настройки центральной панели необходимы удобные средства управления понятные различным категориям пользователей.

При создании системы «Умный дом» большое внимание необходимо уделить построению ее модели, которая позволит:

– продемонстрировать функционирование системы заказчику и

[4] Работа выполнена при поддержке РФФИ (проект № 12-07-00763-а)

оценить ее стоимость на ранних стадиях проектирования;
- определить возможность интеграции компонентов, созданных различными производителями, и провести анализ влияния различных подсистем друг на друга;
- разработать пользовательский интерфейс центральной панели.

Поскольку владелец дома сам определяет набор необходимых ему компонентов, то для каждого конкретного случая требуется выполнять построение уникальной модели системы, которая в дальнейшем должна быть отражена в пользовательском интерфейсе центральной панели. Для этих целей может быть использован предметно-ориентированный язык описания моделей для системы «Умный дом» (SHDL, Smart House Description Language).

Предметно-ориентированный язык (DSL) – это язык моделирования, предназначенный для решения определенного класса задач в конкретной предметной области. В отличие от языков моделирования общего назначения, DSL более выразительны, просты в применении и понятны различным категориям специалистов, поскольку они оперируют привычными для них терминами предметной области. Именно поэтому в настоящее время разработано большое число предметно-ориентированных языков, применяемых для создания систем разного назначения: систем искусственного интеллекта, распределенных систем, мобильных приложений, систем имитационного моделирования и др. [1]. Для поддержки процесса разработки и сопровождения DSL используется специальный вид программного обеспечения, получивший название *DSM-платформа* [3]. Система MetaLanguage предназначена для создания визуальных динамически настраиваемых предметно-ориентированных языков моделирования [2]. Опишем с помощью системы MetaLanguage язык SHDL.

На первом этапе необходимо описать модель языка моделирования (метамодель): определить основные конструкции (сущности) и связи между ними (отношения).

Прежде чем приступить к построению метамодели языка, проведем анализ компонентов, которые могут входить в состав систем «Умный дом». Основными элементами систем такого типа являются:
- системы жизнеобеспечения: отопления, кондиционирования и вентиляции, освещения, безопасности;
- датчики (приборы, отвечающие за снятие различных показаний и передачу их центральной панели): движения, протечки, огня и дыма, закрытия/открытия объекта;
- средства управления системой: голосовое управление, дистанционное управление (с удаленного компьютера, с телефона, с

помощью пульта и др.), сенсорное управление (управление непосредственно с помощью сенсорного экрана центральной панели);
- центральная панель, отвечающая за сбор данных с датчиков, управление системами жизнеобеспечения и получение команд от пользователя.

Основываясь на перечисленных элементах системы «Умный дом», опишем сущности метамодели языка SHDL (см. рис. 1).

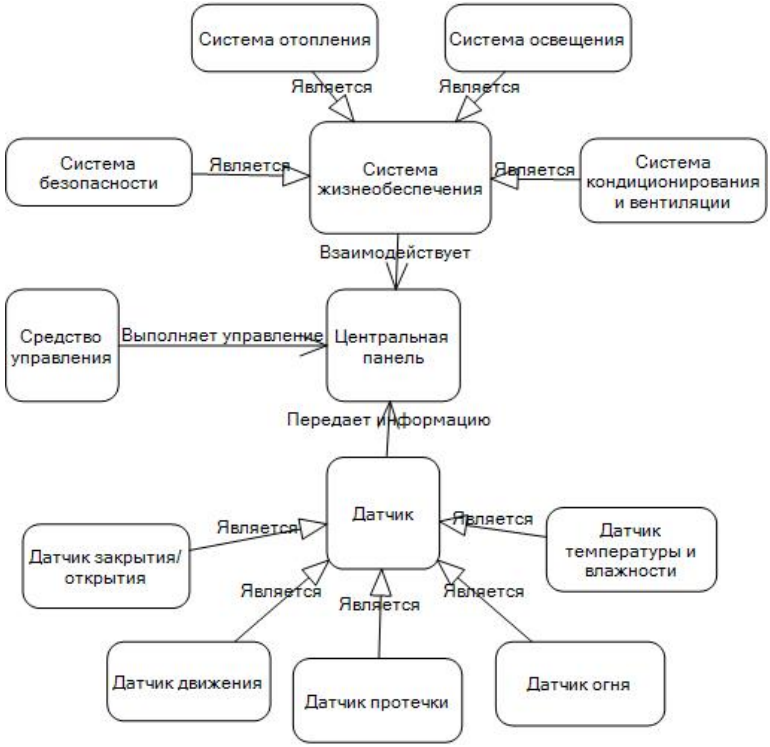


Рис. 1. Метамодель языка SHDL, построенная в системе MetaLanguage

Для унифицированного описания сущностей, соответствующих различным системам жизнеобеспечения, определим абстрактную сущность «Система жизнеобеспечения», которая имеет следующие атрибуты: «Название», «Производитель», «Стоимость», «Объем потребляе-

мой электроэнергии», «Состояние» (определяет активность системы в текущий момент времени). Дочерними для сущности «Система жизнеобеспечения» являются сущности: «Система отопления», «Система кондиционирования и вентиляции», «Система освещение», «Система безопасности».

Сущности «Система отопления» и «Система кондиционирования и вентиляции», помимо наследуемых атрибутов, имеют собственный атрибут «Температура», содержащий значение температуры, которую необходимо поддерживать в помещении. Сущность «Система освещения» также имеет собственные атрибуты «Уровень освещенности», «Источники света». Сущность «Система безопасности» включает в себя систему защиты от протечек и возгораний, систему автоматического пожаротушения, систему охранно-пожарной сигнализации и систему видеонаблюдения. Данная сущность, помимо наследуемых от сущности «Система жизнеобеспечения» атрибутов, имеет собственный атрибут «Состояние безопасности», который может принимать одно из двух значений: «Присутствует нарушение безопасности»/«Нарушений безопасности нет».

Абстрактная сущность «Датчик» является родительской для сущностей, соответствующих всем видам датчиков системы. Она имеет следующие атрибуты: «Название», «Производитель», «Стоимость», «Паспортный номер».

Сущности «Датчик движения» и «Датчик закрытия/открытия», помимо наследуемых атрибутов, имеют свой собственный атрибут «Состояние», который определяет наличие движения в помещении, открытие/закрытие объекта.

Сущность «Датчик протечки» соответствует датчику, который предназначен для фиксации аварийных ситуаций связанных с протечкой воды, газа. Данная сущность имеет собственный атрибут «Уровень давления».

Сущность «Датчик огня», помимо родительских атрибутов, имеет свой собственный атрибут «Чувствительность», который определяет уровень чувствительности датчика к задымлению и изменению температуры.

Сущность «Датчик температуры и влажности» соответствует прибору, отвечающему за снятие показаний температуры и уровня влажности в помещении. Данная сущность имеет собственные атрибуты «Температура» и «Уровень влажности».

Сущность «Средство управления» определяет одно из устройств, позволяющих пользователю отправлять команды центральной панели

управления и контролировать работу системы в целом. Данная сущность имеет следующие атрибуты: «Название», «Вид средства управления» (дистанционное, голосовое, сенсорная панель), «Производитель», «Стоимость».

Сущность «Центральная панель» описывает центральный элемент системы «Умный дом», в него поступает вся необходимая информация с датчиков, и в нем находится центр управления всеми компонентами системы. Именно с центральной панелью пользователь взаимодействует через «Средство управления». Атрибутами этой сущности являются «Производитель», «Стоимость», «Компоненты системы».

После описания всех сущностей метамодели следует перейти к заданию отношений между ними. Метамодель содержит следующие отношения ассоциации:

- «Передает информацию» – это однонаправленное отношение, соединяющее абстрактную сущность «Датчик» и конкретную сущность «Центральная панель».
- «Взаимодействует» – двунаправленное отношение, описывающее взаимодействие абстрактной сущности «Система жизнеобеспечения» и конкретной сущности «Центральная панель».
- «Выполняет управление» – однонаправленное отношение, соединяющее сущности «Средство управления» и «Центральная панель».

Помимо ассоциаций, метамодель содержит девять отношений наследования «Является».

На рис. 2 изображена одна из множества возможных моделей системы «Умный дом», построенная с использованием языка SHDL в системе MetaLanguage.

Заключение

Созданный язык SHDL является достаточно простым и удобным средством описания моделей системы «Умный дом», которое может быть использовано как на этапах проектирования системы в целом и оценки ее стоимости, так и при проектировании пользовательского интерфейса центральной панели, которая позволяет владельцу дома осуществлять контроль за всеми элементами системы и выполнять настройку системы в соответствии со своими потребностями.

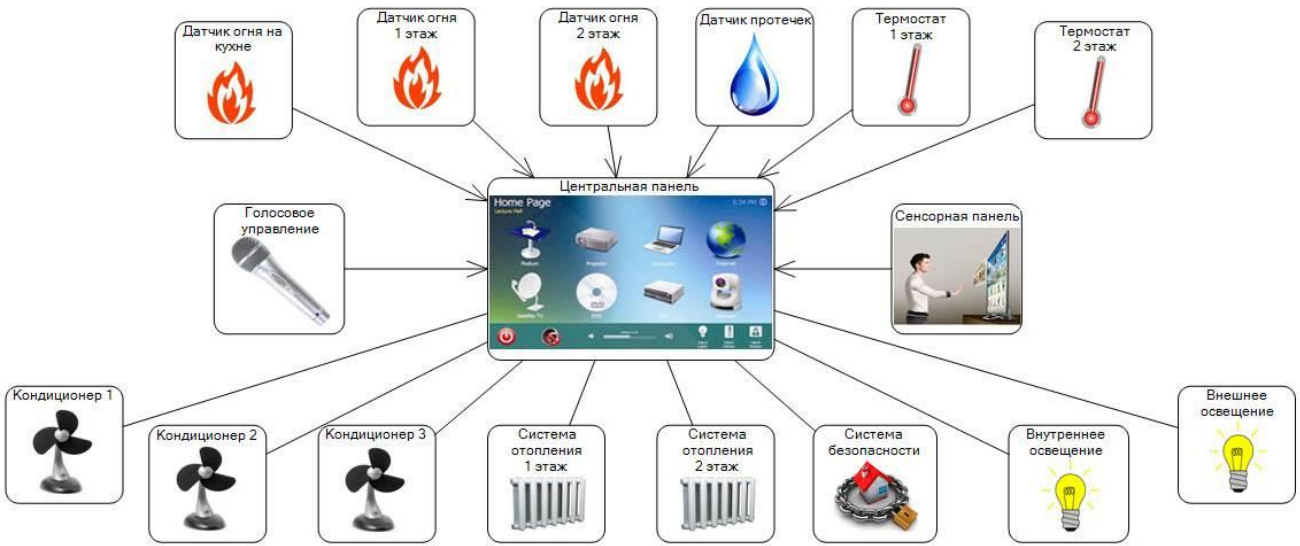


Рис. 2. Модель системы «Умный дом», описанная на языке SHDL

Библиографический список

1. *Лядова Л.Н., Сухов А.О.* Визуальные языки и языковые инструментарии: методы и средства реализации // Труды межд. научно-технической конференции «Интеллектуальные системы» (AIS'10). М.: Физматлит, 2010. Т. 1. С. 374-382.

2. *Сухов А.О.* Инструментальные средства создания визуальных предметно-ориентированных языков моделирования // Фундаментальные исследования. 2013. № 4 (ч. 4). С. 848-852.

3. *Сухов А.О.* Сравнение систем разработки визуальных предметно-ориентированных языков // Математика программных систем: межвуз. сб. науч. ст. Пермь: Изд-во Перм. гос. нац. исслед. ун-та, 2012. Вып. 9. С. 84-111.

Zl.D. Zlatev, V.I. Nedeva

Trakia University (Yambol, Bulgaria)

zlatin.zlatev@trakia-uni.bg; veselina.nedeva@gmail.com

ANALYSIS OF THE APPLICABILITY OF A COMPUTER VISION SYSTEM FOR ASSESSMENT OF THE QUALITY OF QUAIL EGGS

Introduction

In the past years the Japanese quails were used as lab birds because of their easy breeding, early sexual maturity, short reproductive cycle and high egg productivity. The quails are known as a meat and egg source [10]. The eggs of different birds have the same nutritional composition and have the same potential to be used as a food source. On the other hand the information about the quality of the eggs and their use as a food source is essentially limited to chicken eggs [3]. The chicken eggs are very well-studied in regard to their quality, but data about the egg quality of other kinds of bird species is missing in the literature. For the consumer of importance are both the external traits, such as – egg cleanness, freshness, weight, and eggshell thickness, as well as the internal traits – quality of the yolk and egg white, egg components ration and chemical composition [13]. These characteristic are of importance for the industrialized quail farming.

The egg is a biological structure intended for reproduction. It protects and provides a complete diet for the developing embryo, and serves as the principal source of food for the first few days of the chick's life. The Japanese quails are tough birds, which are easy to take care for in a cage. They are resistant to a few diseases that are typical for other types of poultry. The quails lay eggs every 50 days. Under proper conditions these birds can lay up to 200 eggs per year. The life expectation of a quail is around 2.5 years. The quail eggs are white with brown spots; sometimes the eggshell is even tan or off-white coloured. Each bird lays eggs with a specific characteristic eggshell colour [14].

Eggs are classified as a natural food source. In the past years there has been an increased interest in the use of quails for meat and eggs along with

other poultry products. Also there is an increased interest in the industrialized quail farming for meat and eggs. The industrialized farming of these birds leads to the question of the meat and egg quality. The external and internal traits of the eggs and their storage time are quality indicators. The eggs are used as a diet food since they are protein and nutrients source. The assessment of the egg quality depends on their weight, shell strength, ratio between the internal components, such as egg white, yolk, air cell and the possible variations between these traits. Some problems develop during egg storage such as weight loss and deterioration of the quality of whites and yolks. The passage of carbon dioxide and moisture through the eggshell leads to deterioration of the quality of the egg white and yolk and to loss of weight [2, 5].

The technical tools used for assessment of the egg quality are shown in [15]. The up-to-date technical tools use the identification of the following quality indicators:

- discovery of eggshell cracks;
- discovery of blood;
- discovery of leaking eggs.

This study aims at proposing a technical tool using a video sensor and software for assessment of the quail eggs quality using external indicators, because of the increased interest in the industrialized quail farming and egg consumption.

Material and Methods

Hundred (100) eggs, laid by Japanese quail eggs, which were 15 weeks old, and kept in cages with 1 male and 2 female quails in a farm at the village of Bolyarovo, Yambol municipality, were used. The weight of the eggs was determined using the analytical electronic weight Boeco BBI-31 with an accuracy of 0,01 g. The width and length of the eggs was also recorded.

Computer Vision System

For the purpose of the study a computer vision system, with the following components, was used (Fig. 1):

- Video camera – the camera is mounted on a flexible arm, which allows for changing the shooting height. The images are taken at a height of 20 cm.
- Lightning system – the lightning system allows for adjusting the light and removal of the noises. White-light LEDs, with a wavelength of 450 nm, were used.
- A PC with installed software for recognition and image processing using texture traits. The work algorithm of the proposed system is given in Table 1.

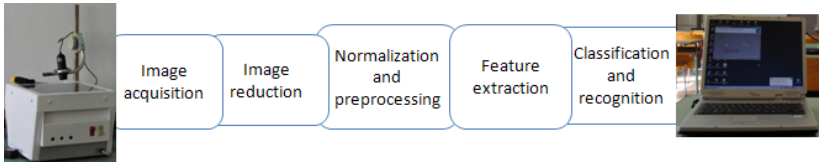


Fig.1. Block-diagram of the computer vision system

Table 1. Algorithm for recognition using texture traits

Step	Work stage	Description
A	*Loading of the original image*	Location and segmentation of the object
B	*Removal of the changes in the contrast*	Determining the image gradient and segmentation threshold
C	*Creation of a binary mask*	Contains the basic elements of the sample
D	*Image filtering*	Using a disk type operators
E	*Setting up the threshold value*	Obtaining a binary mask using settings parameter and an edge operator application
F	*Filling the existing gaps in the closed outline*	Filling of small areas enclosed in the general outline using function and operators
G	*Removal of all elements connected to elements outline*	Setting the removal of diagonal links
H	*Improving the object outlook*	Using double smoothing
I	*Introducing the resulting outline to the original image*	Displaying of the outline over the original image using a specific colour
J	*Recognition of the eggshell traits*	–
K	*Calculating the egg parameters*	The parameters are calculated using the equations give in Table 1

Software System for Image Recognition and Processing

Software system for receiving, processing and analysis of images which is based on texture traits was developed.

The accuracy of recognition is regulated by the radius of the averaging filter of the "disk" type and settings parameter "Fudge Factor". The "disk" type filter transmits a square matrix with a side equal to $d = 2 \times radius + 1$. The radius variation is 1, 2, 4, 6, 8 and 10. The settings parameter Fudge Factor is used for setting up the segmentation threshold and varies between 0,1 and 0,9.

The equations, provided in [1], were used for determining the physical parameters of the eggs – length, width, weight, eggshell thickness, egg shape index. The authors have used regression equations, with a regression coefficient of $R^2 = 0,32 \div 0,83$ for the different models with a significance level $P < 0,01$. The equations used for determining the physical parameters of the eggs are shown in Table 2 [1].

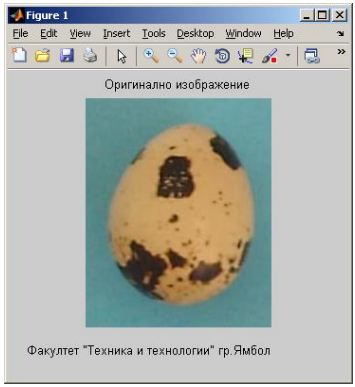
Table 2. Equations used for determining the physical parameters of Japanese quail eggs

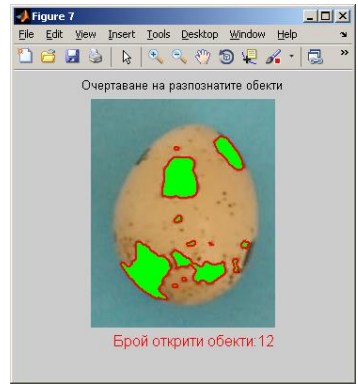
Equation	***Regression coefficient***	***Parameters***	***Source***
Y=-1,658+0,828X_1+0,373X_2	R^2=0,32	Y is predicted egg weight, X_1 – the egg width, X_2 – the egg length	[4]
Y=0,573+0,01532X_3+0,0238X_4	R^2=0,51	Y is predicted eggshell weight, X_3 – egg length, X_4 – egg weight	[7]
Y=0,135+0,0031X_5	R^2=0,54	Y is predicted eggshell thickness, X_5 – egg length	[6]
Y=10,561-0.178X_6–0.045X_7+1,535X_8	R^2=0,99	Y is predicted eggshell surface area, X_6 – egg width, X_7 – egg length, X_8 – egg weight	[6]
Shape index = egg width / egg length × 100			

Results and Discussion

Figures 2 and 3 show the results of the software for image processing, whose function is to separate the egg from the background.

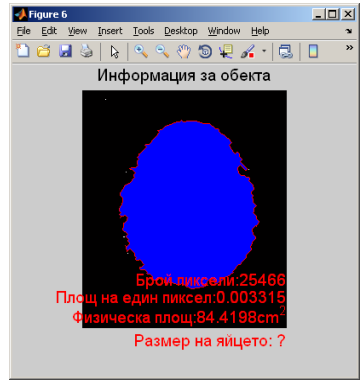
The original image of the egg is loaded in the Matlab work space. After that the object is located and segmented. The segmented object contrasts considerably with the background. The changes in the contrast can be removed with determining the gradient of the image. The gradient and the segmentation threshold are calculated and a binary mask containing the segmented egg is created. The *edge* and *Sobel operators* are used to calculate the threshold values. After that the "Fudge Factor" settings parameter is used for setting up the threshold value and once again the edge function is used to obtain the binary mask containing the segmented egg.





a) Original image **b) Image after processing**

Fig. 2. Detection of eggshell traits (spots)



a) Original image **b) Image after processing**

Fig. 3. Determining the physical parameters of the eggs

The binary mask of the gradient shows the lines with a strong image contrast. These lines do not sufficiently cover/follow the egg outline. If one compares the original image to the processed image, there will be noticeably differences between the lines covering the object in the gradient mask. These differences can be removed by using linear structural elements, which are created using the *strel function*.

The proposed algorithm for recognition and processing of the images of Japanese quail eggs, based on texture traits, is shown in Table 1.

Figure 2 shows the results of the application of the algorithm for recognition of eggshell traits (spots). Camouflage is conferred by background matching and disruption, which are both affected by microhabitat. Microhabitat selection that enhances camouflage has only been demonstrated in species with discrete phenotypic morphs. For most animals, phenotypic variation is continuous. In quail, variation in appearance is particularly obvious in the amount of dark maculation on the light-colored shell. When given a choice, birds consistently selected laying substrates that made visual detection of their egg outline most challenging [8].

The use of the computer vision system for determining the basic physical parameters of the eggs is shown in Fig.3. The efficiency of the developed algorithm was verified by comparing the values, detected by the system, and the values provided by an expert [9].

The obtained results were compared to the requirements, stated in the regulations concerning the quality of the eggs [11, 12].

Conclusion

This article provides a description of a computer vision system, which can be used for determining the quality of Japanese quail eggs using texture traits.

An algorithm was developed for recognition of the eggshell coloring and determining of their physical parameters. The system discovers common defects in the production and storage of the eggs, defects that are given in national and European requirements and standards.

What follows is the development of improvements in the system's work process by modifying the procedures for defect detection with using different classifiers and improving the database for the defects in question.

References

1. *Alkan S., Karabag K., Galic A., Balcioglu M.S.* Predicting Yolk Height, Yolk Width, Albumen Length, Eggshell Weight, Egg Shape Index, Eggshell Thickness, Egg Surface Area of Japanese Quails Using Various Eggs Traits as Regressors // International Journal of Poultry Science. Vol. 7 (1). 2008. ISSN 1682-8356. P.85-88.
2. *Cho J.M., Park Sk., Lee Y.S., Rhee C.* Effect of soy protein isolate coating on egg breakage and quality of eggs during storage // Food Sci. Biotechnology. Vol.11. 2002. P. 392-396.

3. *Dudusola I.O.* Comparative evaluation of internal and external qualities of eggs from quail and guinea fowl // International Research Journal of Plant Science. Vol. 1(5). 2010. P.112-115.
4. *Farooq M., Mian M.A., Ali M., Durrani F.R., Asghar A., Muqarrab A.K.* Egg traits of Fayumi birds under subtropical conditions // Sarhad. J. Agri. Vol.17. 2001. P.141-145.
5. *Genchev A., Mihaylova G., Ribarski S., Pavlov A., Kabakchiev M.* Meat quality and composition in Japanese quails // Trakia Journal of Sciences. Vol.6, № 4. 2008. ISSN 1312-1723 (print), ISSN 1313-3551 (online). P.72-82.
6. *Gulnawaz.* Egg traits and hatching performance of Non-descript Desi chicken produced under backyard conditions in district Charsadda. M.Sc (Hons). Animal Husbandry, PhD Thesis. Department of poultry science, NWFP, Agricultural University Peshawar, Pakistan, 2002.
7. *Khurshid A., Farooq M., Durrani F.R., Sarbiland K., Chand N.* Predicting egg weight, shell weight, shell thickness and hatching chick weight of Japanese quails using various egg traits as regressors // Int. J. Poult. Sci. Vol.2. 2003. P.164-167.
8. *Lovell P.G., Ruxton G.D., Langridge K.V., Spencer K.A.* Egg-Laying Substrate Selection for Optimal Camouflage by Quail // Current Biology. Vol.23, No.3. 2013. P.260-264.
9. *Mladenov M.I.* Intelligent sensors and systems // Ruse, 2010 (original is in Bulgarian language).
10. *Punia B.K., Ramesh B.G., Gnana M.P., Rajasechar A.R.* A study of egg quality traits in Japanese quails // J. Veterinary & Animal Sciences. № 4(6). 2008. P. 227-231.
11. Regulation (EC) № 1028/2006 of 19 June 2006 on marketing standards for eggs (original is in Bulgarian language).
12. Regulation № 1 of 09.01.2008 on the requirements for eggs for consumption (original is in Bulgarian language).
13. *Song K.T., Choi S.H., Oh H.R.* A comparison of Egg Quality of Pheasant, Chukar, Quail and Guinea Fowl. Asian-Aus // J. Anim. Sci. № 13(7). 2000. P.986-990.
14. *Vashin I., Stoyanchev T., Roussev V.* Prevalence of microorganisms of the campylobacter genus in quail (Coturnix Coturnix) eggs // Bulgarian Journal of Veterinary Medicine. Vol.11, No.3. 2008. P.213-216.
15. Site of company "Агроконсулт Инженеринг ЕООД" [URL: http://www.agroconsulteng.com/index.php?option=com_content&view=article&id= 97&Itemid=35].

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В БИЗНЕСЕ, НАУКЕ И ОБРАЗОВАНИИ

О.Л. Викентьева, А.И. Дерябин, Л.В. Шестакова

О ПОДХОДЕ К РАЗРАБОТКЕ МОДЕЛИ ПРОВЕДЕНИЯ ДЕЛОВОЙ ИГРЫ В СТУДИИ КОМПЕТЕНТНОСТНЫХ ДЕЛОВЫХ ИГР

Введение

Сфера образования превращается во все более важную отрасль экономики, производящую профессиональный интеллект [2]. В этих условиях выпускники вузов должны обладать профессиональными компетенциями, т.е. способностью применять знания, умения, навыки и личностные качества для успешной деятельности в профессиональной среде. Подготовка конкурентоспособного выпускника, способного принимать эффективные решения в сфере профессиональной деятельности, требует использования новых образовательных технологий, активных методов обучения. В связи с этим следует отметить все более активное использование деловой игры в учебном процессе. Деловую игру можно рассматривать как моделирование реальной деятельности выпускника в различных производственных ситуациях.

В рамках работ по управлению персоналом значительное внимание уделяется процедурам определения и формирования профессиональных компетенций, обучения персонала [3], а также разработке программных продуктов, реализующих эти процедуры. Существуют европейские проекты, ориентированные на *компетентностное обучение* (*Competency-based education*).

Компетенции, определяемые образовательными и профессиональными стандартами, могут сформироваться только в условиях, близких к производственным. Для имитации таких условий предлагается включить в систему подготовки выпускников (бакалавров) использование модели производственной деятельности, в которой отра-

жены все этапы жизненного цикла предприятия. Такая модель может быть реализована с помощью *компетентностных деловых игр* (КДИ). Рассматривается функциональная модель проведения деловой игры с помощью подсистем *инструментальной среды* «*Студия компетентностных деловых игр*» (СКДИ) [1, 4].

Структура студии компетентностных деловых игр

Структурная схема СКДИ [1] включает в себя ряд подсистем.

Подсистема проектирования предназначена для разработки сценариев деловых игр, моделей предметных областей, на базе которых выполняются сценарии, а также учебно-методических материалов для проведения игр, и контрольно-измерительных материалов.

Подсистема проведения предназначена для проведения деловой игры с использованием материалов, разработанных в подсистеме проектирования. Она представляет собой человеко-машинную (эргатическую) систему.

Подсистема мониторинга предназначена для отслеживания хода игры и результатов игроков.

Подсистема анализа предназначена для обработки результатов, полученных в ходе игры, и получения отчетов о состоянии качества методического обеспечения, ходе деловой игры (ДИ), состоянии игроков.

Подсистема корректировки предназначена для оперативного изменения хода деловой игры и изменения элементов ДИ, разработанных подсистемой проектирования.

Подсистема измерения использует контрольно-измерительные материалы, которые позволяют определить уровень сформированных компетенций.

Подсистемы проведения, мониторинга, анализа и корректировки образуют *контур оперативного управления* проведением ДИ.

Подсистемы проектирования, проведения, анализа и подсистема корректировки образуют *контур управления методическим комплексом* ДИ.

Подсистемы проведения, измерения и мониторинга образуют *контур управления компетенциями*, вырабатываемыми в процессе ДИ.

Наличие информационных контуров обеспечивает высокую управляемость процесса формирования компетенций, а также возможность формирования индивидуальных траекторий обучения.

Бизнес-процессы предметной области (предприятия) представлены в виде ориентированного графа, при прохождении путей которого можно формировать элементарные компетенции. В КДИ используются

графовые модели учебных и управляющих учебных бизнес-процессов. При переходе к моделям учебных бизнес-процессов (УБП) проводится системный анализ и построение онтологии предметной области.

При реализации программного комплекса СКДИ возникает несколько проблем, в частности, проблема представления действий обучаемого и управление этими действиями в соответствии с логикой предметной области для достижения цели – формирования заданного уровня компетенции. Решение данной проблемы связано с разделением КДИ на автоматную и операционную модели. *Автоматная модель* реализует управление формированием компетенций. *Операционная модель* реализует процесс формирования компетенций. Автоматная модель строится с учетом сценария, графа учебного управляющего процесса и онтологии предметной области. При этом выбирается язык, позволяющий реализовать алгоритм управления. Для проектирования операционной модели используется сценарий, граф учебного основного бизнес-процесса и онтология предметной области.

В настоящей работе представляется модель проведения деловой игры, спроектированной с помощью подсистемы проектирования СКДИ.

Модель проведения деловой игры

Основными действующими лицами (*актерами*) являются *разработчик* (эксперт, бизнес-аналитик, программист, специалист по тестированию и др.), *администратор* и сам *игрок*. Внешними сущностями являются *системы хранения*: хранилище ресурсов, хранилище протоколов, хранилище диагностических карт.

Хранилище ресурсов содержит тестовые ресурсы и ресурсы, формирующие контекст сеанса игры. *Хранилище протоколов* обеспечивает хранение протоколов сеансов игры и протоколов итогового тестирования. *Диагностическая карта* игрока содержит информацию о степени усвоения компетенций (по ресурсной базе компетенции, по способам действий, по мотивационно-ценностной составляющей и т.п.).

Рассмотрим *прецеденты* более подробно.

Проведение сеанса игры предполагает, что данная игра, включающая в себя сценарий, игровые и тестовые ресурсы, учебно-методический комплекс (УМК) и т. д., разработана с использованием подсистемы проектирования СКДИ.

Подсистема «*Проведение игры*» включает два прецедента – «Проведение сеанса игры» и «Итоговое тестирование».

Прецедент «*Проведение сеанса игры*» включает следующие шаги:

– игрок выбирает игру на сайте СКДИ;

– игрок выполняет настройку игры, для чего осуществляет:
  - выбор компетенций;
  - выбор режима (одно- или многопользовательский);
  - выбор роли.

Управляющий модуль подсистемы проведения игры в цикле, пока не будет выполнена команда «Выход», должен:

– сформировать и вывести игроку контекст игры;
– выполнить транзакцию, установить значение регистра состояния (PC) операционной модели;
– оценить результат транзакции (в баллах).

Прецедент «*Итоговое тестирование*» выполняется, когда полностью выполнен сценарий деловой игры, сформированы оценки текущего уровня усвоения компетенции. Кроме того, должны быть сформированы тестовые ресурсы для итогового тестирования (контрольно-измерительные материалы, КИМ). Прецедент выполняется игроком, в нем используется хранилище тестовых ресурсов.

В процессе выполнения итогового тестирования управляющий модуль подсистемы проведения игры в цикле, пока не будет выполнена команда «Выход», должен:

– сформировать и вывести игроку тестовый вопрос (задание);
– выполнить тестовую транзакцию, установить значение регистра состояния (PC) операционной модели;
– оценить результат транзакции.

Сбой во время выполнения транзакции повлечет за собой откат транзакции.

Подсистема «*Мониторинг*» включает в себя прецеденты «Ведение протоколов хода игры» и «Ведение протоколов тестирования».

Прецедент «*Ведение протоколов хода игры*» включён в прецедент «Проведение сеанса игры» и заключается в записи результата транзакции в хранилище протоколов о результатах сеанса деловой игры.

Прецедент «*Ведение протоколов тестирования*» включён в прецедент «Итоговое тестирование» и заключается в записи результата в хранилище протоколов тестирования.

Подсистема «*Анализ*» включает прецеденты «Анализ протоколов хода игры» и «Анализ итогового тестирования».

Прецедент «*Анализ протоколов хода игры*» расширяет прецедент «Ведение протоколов хода игры» и осуществляет агрегирование результатов множества транзакций по заданным параметрам (по игроку, по компетенции, по дате и т.п.), а также выполняет анализ результатов транзакций средствами OLAP-анализа и Data Mining.

Прецедент «*Анализ итогового тестирования*» расширяет преце-

дент «Ведение протоколов тестирования» и заключается в агрегировании результатов тестовых транзакций по составляющим компетенции.

Результатами выполнения прецедентов являются отчеты об уровне усвоения компетенций и о качестве итогового теста и учебных бизнес-процессов.

Подсистема «*Измерение*» включает в себя прецедент «Получение результатов», который выполняется после окончания сеанса игры. Данный прецедент включён в прецеденты «Ведение протоколов тестирования» и «Ведение протоколов хода игры». На данном этапе необходимо выполнить следующие шаги:

- получить из результата измерения каждого вопроса оценку его ресурсной структуры (умения и знания);
- построить диагностическую карту игрока.

Подсистема «*Корректировка*» включает прецеденты «Корректировка сеанса игры» и «Корректировка игры».

Прецедент «*Корректировка сеанса игры*» расширяет прецеденты «Анализ протоколов хода игры» и «Проведение сеанса игры».

Прецедент «Корректировка сеанса игры» выполняется в зависимости от текущего уровня усвоения компетенции:

- игра может быть закончена при достижении заданного уровня компетенции;
- можно повторить игру полностью, если получена оценка ниже заданного уровня;
- можно повторить только модуль игры, по которому получена оценка ниже заданного уровня.

Для его выполнения необходимо, чтобы была сформирована оценка текущего уровня усвоения компетенции и заданы значения уровней корректировки игры.

Выполнение прецедента заключается в следующем:

- из подсистемы анализа загружается отчет с оценкой уровня усвоения компетенции;
- если оценка больше заданного уровня корректировки *Kmax*, игра прекращается;
- если оценка меньше заданного уровня корректировки *Kmin*, игра повторяется сначала.

Результатом выполнения прецедента является корректировка текущего сеанса игры.

Прецедент «*Корректировка игры*» предполагает внесение изменений в модели учебных бизнес-процессов (УБП) и итоговый тест.

В рамках выполнения этого прецедента

- выполняется анализ отчетов о качестве моделей УБП и итого-

вого теста;
- формируется техническое задание на модификацию ДИ;
- техническое задание (ТЗ) передаётся в систему проектирования.

Результатом выполнения прецедента является ТЗ на модификацию ДИ.

Заключение

В работе описаны основные прецеденты подсистемы проведения игры в СКДИ, обеспечивающие проведение сеанса игры, итоговое тестирование, ведение протоколов хода игры, ведение протоколов тестирования, анализ протоколов хода игры и хода тестирования, получение результатов, корректировку сеанса игры, корректировку игры. Выполнена детализация прецедентов с указанием основных действующих лиц и подсистем хранения. Результаты выполненного анализа предлагается использовать для проектирования детальной архитектуры подсистемы проведения игры в СКДИ и реализации её компонентов.

Библиографический список

1. *Викентьева О.Л., Дерябин А.И., Шестакова Л.В.* Концепция студии компетентностных деловых игр [Электронный ресурс] // Современные проблемы науки и образования. – 2013. – № 2; URL: http://www.science-education.ru/108-8746 (дата обращения: 03.04.2013).
2. *Викентьева О.Л., Дерябин А.И., Шестакова Л.В.* Формирование компетенций согласно ФГОС ВПО третьего поколения на примере подготовки бакалавров бизнес-информатики // Рождественские чтения: материалы XVII Всероссийской науч.-метод. конф. по вопросам применения ИКТ в образовании, 10-11 января 2013. Пермь, 2013. Вып. 17. С. 16-19.
3. *Крюков К.В., Кузнецов О.П., Суховеров В.С.* О понятии формальной компетентности научных сотрудников // OSTIS-2013: Материалы III международной научно-технической конференции. Минск : БГУИР, 2013. С. 143-146.
4. *Deryabin A., Shestakova L., Vikentyeva O.* The construction of competency-based business game operational model // International Journal «Information Technologies & Knowledge», Vol.7, No.:4, 2013. P. 303-313.

Н. Петров, Е. Димова

ЦЕНА ТОВАРА КАК ОБРАТНАЯ ИНФОРМАЦИОННАЯ СВЯЗЬ МЕЖДУ ПРОИЗВОДИТЕЛЕМ И КЛИЕНТОМ

Технический университет (София),
Национальный исследовательский Тракийский университет
(Стара Загора)

nikipetrov_1953@abv.bg; emdimova@abv.bg

Введение

Философ – классик теории капитализма – Карл Маркс даёт следующее определение цены: «Выражение стоимости товара в золоте (*X* товара *A* = *У* денежный товар) – это его денежная форма, т.е. его стоимость» [1]. Изучая процесс обмена труда и товаров, Маркс сформулировал *Закон стоимости* следующим образом: «*Люди не обмениваются товаром произвольно, а всегда стремятся к эквивалентности*. Это означает, что труд в одной форме обменивается на то же самое количество труда в другой форме. В этом суть Закона о стоимости.» [1–4].

Однако известно, что рыночные цены на отдельные товары отклоняются от их стоимости. Закон стоимости действует как стихийный регулятор, именно через механизм цен. В то же время подключается и механизм спроса и предложения. Механизм, который регулирует товарное производство, исследуется экономической наукой уже более двух столетий. Ответ на этот вопрос является особенно важным для нашего исследования, так как мы рассматриваем общественное производство как динамическую самоорганизующуюся систему.

Во-первых, мы должны ответить на вопрос: «Почему цена товара отклоняется от стоимости товара?».

Во-вторых, нас интересует вопрос: «Почему цена не совпадает со стоимостью товара, учитывая факт, что она является одной из форм

проявления стоимости?». Несомненно, причину этого следует искать в некоторых дополнительных факторах.

Почему существуют отклонения цен товаров от их реальной стоимости?

Анализ отношений стоимости на сегодняшний день показывает, что трансформация индивидуального труда в общественно необходимый труд представляет собой процесс формирования стоимости. Если предположить, что формирование стоимости происходит в денежной форме, то отклонение индивидуальных денежных расходов от средних общественно необходимых расходов не является отклонением цены от стоимости. Это отклонение стоимости отдельного случая от среднего арифметического. Вообще вопрос не в отклонениях индивидуальных цен от рыночных. Нас интересует проблема: «Почему существуют отклонения цен товаров от их реальной стоимости (измеренной в золоте)?».

Чтобы решить возникший казус авторы рассматривают следующую *аксиому цен*: Общая сумма цен товаров равна общей сумме стоимостей всего товарного рынка.

Как следствие аксиомы цен, поставленный выше вопрос сформулирован следующим образом: «Почему существует отклонение индивидуальной цены от индивидуальной стоимости товара или отклонение рыночной цены от рыночной стоимости товара?». Конечно же, эти отклонения – в пределах стоимости гомеостаза соответствующей страны или города определенной страны. Отклонение цены от стоимости связано с воздействием потребительской стоимости товара на ценообразование. Если мы принимаем, что товар имеет две стороны, два фактора, которые определяют его – *потребительская стоимость* и *реальная стоимость*, то на отклонение цены от ее номинальной стоимости, оказывают важное влияние оба фактора. В некоторых случаях отклонение цены от стоимости товара определяется потребительской стоимостью, а в других – формирование потребительской стоимости товара зависит от его реальной стоимости. Таким, например, является случай, когда материал более высокого качества заменяется другим, менее дорогим материалом низкого качества. Часто потребительская стоимость товара снижается из-за стремления сделать так, чтобы он был дешевле и стал более доступным для покупателя.

Влияние потребительской стоимости на цену происходит на рынке через *механизм спроса и предложения*. Соотношение между спросом и предложением выражает количественные соотношения в распределении общественного труда в различных отраслях экономической

жизни определенной страны. Предполагая, что все товары в данной отрасли производятся при средних для общества условиях интенсивности и производительности труда, то есть, если исключить любые отклонения индивидуальной стоимости от стоимости отдельных товаров, то в данном случае можно согласиться с утверждением: «Цена этих товаров – ниже их реальной стоимости» [3].

Что является причиной отклонения от реальной цены?

Общественный спрос устанавливает границы, в пределах которых распределяется общественный труд. Возможно, часть общественного труда, затраченного в определённом секторе, не была общественно необходимой и использовалась в избытке. Это, однако, проявляется на рынке как отклонение цены от стоимости товара. Здесь следует отметить, что стоимость включает в себя часть труда, которая не является общественно необходимой. Он включён в стоимость, поскольку была создана потребительская стоимость. При этом на общественном уровне эта потребительская стоимость не реализуется в потреблении и оказывается, что она в избытке.

В этом случае Закон стоимости Маркса действует путем снижения цены ниже реальной стоимости затрат труда, тем самым расширяя возможности для потребления, то есть потребителям предлагается признать общественно необходимым и тот труд, который затрачен в избытке.

Дисбаланс количества товаров сверх необходимости, восстанавливается противоположным образом. Так, восстановленное равновесие имеет временный характер и выражает дисбаланс в распределении труда между различными секторами экономики. Когда в одной отрасли производят товары в избытке, а в другой отрасли производят товаров меньше потребности, это приводит к повышенному спросу последних (дефицитных) товаров. В результате цены на товары с высоким спросом поднимаются искусственным путём выше их реальной стоимости. При этом рост цены продукта снижает спрос на него со стороны потребителей. Таким образом, согласно Закону стоимости Маркса соответствие между производством и потреблением восстанавливается на рынке в определённых пределах. Этот закон является регулятором диспропорции существующего производства в каждой стране, если, конечно, ее «капитализм» не спекулятивен и соблюдаются самые общие принципы рынка. Закон стоимости действует как стихийный регулятор пропорций между производством и потреблением, а также пропорций в разделении труда между различными отраслями экономики.

Главное средство, через которое действует Закон стоимости, – это *цена товара* (материального или нематериального). Здесь термин «нематериальный товар» означает продукты интеллектуального труда – литературу (книги, поэзию, статьи и т.п.), произведения других жанров искусства (музыку, живопись и пр.) и т.д. Конечно, товаром являются и деньги, находящиеся на хранении в виде банковского вклада. Увеличение стоимости этого товара (депозита) за соответствующий интервал времени является компенсацией за существующий риск банкротства банка или снижения ликвидности денег [5]. При этом будущая стоимость денег определяется увеличением суммы процентов за период вклада. В соответствии с политикой банка обычно используются сложная процентная ставка или простая процентная ставка.

Сложная процентная ставка является процентной ставкой, которая начисляется за каждый наблюдаемый интервал времени к сумме вложенного капитала плюс проценты за предыдущий (априори) интервал времени.

Расчёт будущей стоимости с использованием сложных процентов осуществляется по следующей формуле:

$$FV = C_0 \times (1 + r)^n, \quad (1)$$

где FV – будущая стоимость (Future Value) денег; C_0 – сумма вложенных денег, инвестированного в настоящий момент капитал; r – процент банка (к сожалению с ним возможны спекуляции); $(1 + r)^n$ – сложная процентная ставка; n – количество лет, за которые вложены деньги [5].

Когда деньги вкладываются при *простой процентной ставке*, будущая стоимость определяется по формуле

$$FV = C_0 \times (1 + r \times n). \quad (2)$$

Будущая стоимость капитала, инвестированного на срок менее одного года, вычисляется по формуле

$$FV^{nm} = C_0 \times (1 + r/m)^{nm}, \quad (3)$$

где r – годовая процентная ставка банка; m – количество периодов в году; n – количество лет, на которые капитал инвестируется; r/m – ставка за определённый период в году [5].

Из формул (1)–(3) следует, что стоимость (цена) капитала – это его информационный сигнал о динамике рынка.

Цена как информационный сигнал на рынке

Цена – это *информационный сигнал*, который товаропроизводители получают от рынка (свободного или регулируемого властями соответствующей страны) [3, 4]. По этому сигналу товаропроизводители получают информацию о реализации своего труда, т.е. об обществен-

ном признании их индивидуального (частного или коллективного) труда. Таким образом, цена функционирует в качестве обратной информационной связи между производителями и потребителями, между реальной стоимостью товара и его потребительской стоимостью. Это кибернетическое взаимодействие показано на рис. 1.

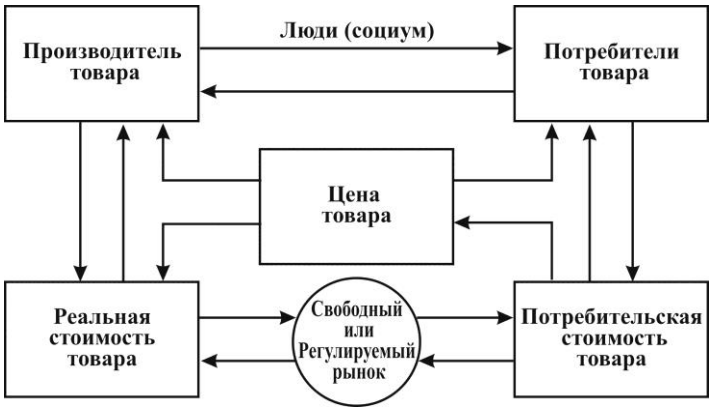


Рис. 1. Кибернетическое взаимодействие между людьми (социумом) – производителями и потребителями, ценой товара (реальной и потребительской) и рынком – свободным (капиталистическим) или регулируемым (социалистическим)

Схема, приведённая на рис. 1, показывает, что цена товара является отрицательной обратной связью, которая, с одной стороны, приводит к соответствию потребительскую стоимость и спрос, а с другой стороны, корригирует количественный предел необходимой потребительской стоимости, которая должна быть произведена при средних для общества условиях.

Проведённый краткий анализ кибернетического взаимодействия, показанного на рис. 1, даёт основания авторам вывести теорему о цене относительно надёжности производства.

Ценовая теорема надёжности производства: Цена действует как негативный фактор отрицательной информационной обратной связи в кибернетической системе производителей и потребителей. Производитель или компания, которые создают высококачественные товары за меньший срок, получают цену ниже их реальной стоимости. Незавидна участь тех, кто не в состоянии реагировать на рыночные условия или производят товары за больший срок, чем среднее необходимое на это время.

О производстве, цене и доходах населения

После анализа, благодаря которому получена ценовая теорема надёжности производства, следует обратить внимание на исследование вопроса об общественном спросе при товарном производстве и доходах населения. В этом случае цена товара играет роль *предварительной оценки потребительских стоимостей* как обратная информационная зависимость между потребностью общества и количеством общественного труда. Здесь цена носит характер регулирующего механизма потребительской стоимости. Кроме того, она является носителем регулирующего механизма в области стоимости. Общественное производство регулируется через отклонения цен от реальной стоимости. Она регулирует пропорции стоимости, а также пропорции в распределении конкретного труда общественного потребления и общественных потребностей. Схематически это показано на рис. 2 [3].

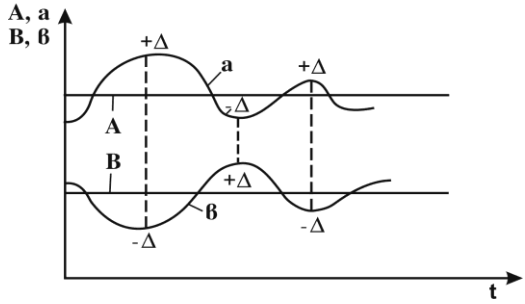


Рис. 2. Цена как регулирующий механизм потребительской стоимости и стоимости

На приведённом графике (рис. 2) использованы следующие обозначения: *A* – стоимость товара при среднем общественно необходимом времени производства; *a* – цена товара; *B* – общественная потребность, выраженная как определённый платёжеспособный спрос; *в* – количество потребительских стоимостей. Между двумя кривыми – между количеством произведенных товаров и отклонением цены от стоимости – обозначена пунктиром обратная отрицательная связь.

Связанными являются плюс (+) с минусом (–), так что динамика приводит к обратному отклонению: плюс (+) с тенденцией к минусу (–) и минус (–) с тенденцией к плюсу (+). Абсолютная величина отклонений дает нам представление об энтропии в общественном производстве.

Отклонения всегда балансируют и дают гомеостаз системе. Термин «гомеостаз» был введен в 1933 г. Уолтером Кенноном (экономистом США), и обозначает свойство открытой системы контроля для регулирования внутренней среды с тем, чтобы сохранить ее устойчивое состояние динамического равновесия, используя управление внутренними или дополнительными экономическими регуляторами [6]. Таким образом, через непропорциональное развитие осуществляются пропорции в товарном хозяйстве. Это выражает постоянные отклонения гомеостаза системы, соответствующие объективному процессу изменений на рынке. При этом следует учитывать тот факт, что общая сумма рыночных цен равна сумме производственных цен, соответственно, стоимости цен.

То же самое относится и к линии *B*, так как при соответствующих производственных мощностях и рабочей силе существует идеальная возможность производить то, что нужно.

Предполагается, что абсолютная стоимость – плюс (+), соответственно, минус (–) является потерей для общества, что проявляется в виде потери стоимостного количества материально-энергетического богатства. Это означает, что чем выше энтропия, тем больше потери общества в стоимости или в потребительской стоимости. Общественная производительность труда тем больше, чем меньше энтропия в общественном производстве [6]. В этой ситуации есть возможность для изучения роли *Закона стоимости* как стихийного регулятора при капитализме, то есть для изучения информационного механизма, посредством которого осуществляется самоорганизация производства. Это осуществляется путем *прямой отрицательной обратной связи* между движением сигналов стоимости и сигналов потребительской стоимости.

Когда количество производимых товаров на рынке меньше, чем спрос, рыночная цена поднимается выше стоимости. Таким образом, работает механизм саморегулирования путём реализации большей прибыли, чем в среднем. При этом начинает действовать *сигнал о норме прибыли* и стимулируется производитель (капиталист), который начинает увеличивать производство. Как только количество товара превышает спрос на рынке, падает прибыль производителя ниже средней за соответствующий период. Этот информационный сигнал свободного капиталистического рынка заставляет производителя снизить свои затраты на производство (например, сократить персонал), повысить качество производства, перебросить капитал в производство другого типа или же сделать что-либо экономически рациональное. Известно, что капиталистическое чутье предпринимателей является

наиболее чувствительным к сигналу о прибыли.

Таким образом, Закон стоимости Маркса регулирует движение потребительской стоимости, т.е. материально-энергетические пропорции капиталистического хозяйства.

Предположим, что получится противоположный эффект потребительской стоимости. Это тот случай, когда появляются новые потребности, удовлетворение которых с точки зрения капиталиста обозначает увеличение прибыли. Тогда естественным образом увеличивается интенсивность производства, соответственно, создаются новые рабочие места и повышается уровень доходов.

Однако мы должны отметить некоторые важные моменты в процессе саморегулирования капиталистического производства.

Во-первых, саморегулирование капиталистического производства не терпит больших отклонений, т.е. не терпит энтропии. Это означает, что спонтанно, независимо от сознания общества в целом, действие Закона стоимости стимулирует рост общественного богатства. Уменьшение энтропии капиталистического производства приводит к улучшению его организации.

Во-вторых, отдельные капиталисты (современные бизнесмены) могут стать богатыми или закончить банкротством (таким, как банкротство большой части банков на Уолл-стрит за период 2005–2010-х), в то время как рабочие могут только иметь или не иметь работу (заработную плату). Исходя из этого, граница их обогащения (или, соответственно, обнищания) – только в пределах фонда заработной платы.

В-третьих, при капитализме процесс производства имеет определенную форму, особые информационные сигналы и соответствующий механизм саморегулирования. В то время как производство при социализме сохраняет лишь некоторые особенности этого процесса.

Формы информационных сигналов изменяются, а также меняются и информационные связи, т.е. механизм саморегулирования производства подчинён тенденции надёжности общества.

Заключение

В результате информационно-экономического анализа, основанного на изучении цен товаров и производственно-потребительских отношений, авторы пришли к шести основным выводам:

1. Существует корреляция между регуляторными механизмами по обеим сторонам стоимости товаров – потребительская стоимость и реальная стоимость. При этом нарушение гомеостаза по одной стороне возмещает гомеостаз по другой. Авторами было доказано через анализ

кибернетической модели (схема приведена на рис. 1), что между ними есть отрицательная обратная связь, осуществляемая ценой товара.

2. Каждая энтропия стоимости производства и потребления является потерей для общества. Следовательно, нельзя допускать большого роста цен и заработной платы. Поэтому не случайно заявление Карла Маркса, что соотношение между максимальной и минимальной заработной платой в товарном производстве, независимо от того, является рынок свободным или регулируемым, не должно быть более десяти.

3. Регулирующий механизм стоимости построен по иерархическому принципу, как в любой системе управления. Иерархия механизма стоимости объясняется тем, что объективно величина стоимости товара формируется на различных уровнях. Кроме того, стоимость на каждом верхнем уровне является критерием для измерения и сравнения стоимости товаров на нижних уровнях.

4. Регулирующий механизм стоимости сохраняет и восстанавливает гомеостаз экономической системы управления обществом, обеспечивая эквивалентность в ценовых отношениях. Любое нарушение ценовой эквивалентности приводит к увеличению энтропии в целом.

Несмотря на то, что в системе цен любые расхождения в плюс и минус уравновешивают друг друга, дисбаланс в отдельных отношениях искажает основную пропорцию между платежеспособным спросом и стоимостью товаров, предлагаемых к продаже.

5. Восстановление стоимости и самостоятельный рост отдельных элементов товара имеют огромное значение для практики и управления хозяйственными процессами.

Элементы общественного производства восстанавливаются материально-энергетически только в рамках общественного воспроизводства. При этом механизм регулирования цен принудительно восстанавливает баланс как в масштабе всего общества, так и относительно каждого отдельного товара.

6. Общественное производство является открытой системой управления, которая начинается с природы, с промышленности, через которую возмещается сырье и проходит через последовательность процессов создания потребительских стоимостей. При этом любой момент создания потребительской стоимости в то же время является моментом потребления другой потребительской стоимости.

Это означает, что материально-энергетическое восстановление любой потребительской стоимости должно пройти через всю открытую цепь – через природу, процесс производства, рекламу и конечное потребление товаров населением.

Библиографический список

1. *Маркс К.* Капиталът. Том 1. София: БКП, 1948.
2. *Маркс К.* Капиталът. Том 3. София: БКП, 1953.
3. *Николов Ив.* Кибернетика и икономика. София: Наука и изкуство, 1971.
4. *Кронрод Я.* Закон стоимости и экономика капитализма и социализма // Мировая экономика и международные отношения. 1967. № 1. С. 57-70.
5. *Петров Г.* 14 урока по фирмени финанси. Стара Загора: Тракия, 2001.
6. *Петров Н.* Модели за управление на технико-икономически системи: учебник за студенти. София: ТУ, Бургас: ЕЦНОК, 2008.

Е.В. Поздеева, С.А. Стринюк

Национальный исследовательский университет
«Высшая школа экономики»
(Пермский филиал)

epozdeeva@hse.ru; strinuk@mail.ru

СОЗДАНИЕ АНГЛОЯЗЫЧНОЙ СРЕДЫ ПРИ ОБУЧЕНИИ АНГЛИЙСКОМУ ЯЗЫКУ: ПРОБЛЕМЫ И ПУТИ ИХ РЕШЕНИЯ

Введение

Важной и неотъемлемой частью процесса овладения английским языком является создание языковой среды. Под этим понимается создание условий, которые способствуют более быстрому и качественному освоению всех видов речевой деятельности, необходимых для осуществления успешной коммуникации на иностранном языке. Мы сознательно употребляем термины «овладение» и «освоение», а не «изучение», поскольку под последним мы понимаем некий академический, умозрительный, интерес к иностранному языку как таковому, в отличие от умения пользоваться им на практике в ситуациях реального общения.

В последнее время о необходимости создания языковой среды говорилось достаточно много. В статьях и исследованиях предложены многочисленные способы использования Интернет-ресурсов в целом, обучающих ресурсов, обучения отдельным аспектам языка либо применяя какую-либо технологию (см. об этом, к примеру: [1]).

Все исследования направлены на поиск эффективных методических решений для проведения более качественного урока (читай: нескучного, с применением ИКТ и т.д.) по сути, подменяя изучение иностранного языка с помощью создания иноязычной среды использованием некой суммы технических приемов и средств использования ИКТ на уроке.

Еще одним определенным недостатком или, скорее, упущением со стороны исследователей является то, что они, как правило, говорят о создании иноязычной среды на *уроке*, в то время как создание иноязычной среды во *внеурочное* время не воспринимается как важная часть работы преподавателя. Общепринятое мнение заключается в том, что создание внеурочной иноязычной среды может быть успешным лишь вне привычного языкового окружения учащегося. Другими словами, погрузиться в нее можно лишь, покинув Россию и переехав жить или учиться в англоязычную страну. Таким образом, учащийся оказывается в среде, где он вынужден использовать английский язык как средство общения, получения и передачи информации. Преподаватель, согласно этой концепции, не может влиять ни на создание, ни на выбор внеурочной языковой среды учащегося.

Нам подобная практика представляется, как минимум, неразумной. В лучшем случае она приводит студентов к неуверенности в своих способностях, в худшем – к твердому убеждению в неспособности заговорить на английском языке. Очень часто студенты теряют всякий интерес к изучению английского языка в университете (мы сейчас говорим только о нелингвистических специальностях) именно потому, что не видят ценности приобретаемых практических навыков, поскольку не имеют возможности применить их *вне урока*, или же недостаточно мотивированы искать подобные возможности и пользоваться ими. Отсутствие языкового окружения, как и отсутствие желания искусственно его моделировать, значительно сужают границы сферы деятельности студентов, снижают их коммуникативные потребности, заставляют воспринимать навыки в различных видах деятельности как не связанные между собой и ведут к неумению использовать их в реальной ситуации общения. В результате студенты способны (более или менее качественно) решать задачу в узко-организованной среде (например, выполнять задания на выбор необходимой грамматической формы на уроке), но при этом не переносят этот навык в реальную жизнь. Это аналогично тому, как, зная таблицу умножения и используя ее для решения арифметических задач в классе, учащийся не применяет эти знания, делая покупки в магазине. Именно поэтому студенты начинают воспринимать английский язык только как учебный предмет (часто сложный и скучный), как непреодолимое препятствием на пути к получению стипендии или (к счастью, редко) как вполне реальную угрозу своему пребыванию в рядах студентов вуза.

Между тем, современная информационная среда предоставляет обширные возможности перейти на качественно иной уровень владения иностранным языком, не проводя месяцы или годы в другой

стране. Развиваться в пределах профессиональных интересов студенты могут, используя он-лайн-курсы по интересующим специальностям, читая статьи из библиотечных ресурсов англоязычной периодики; из развлечений – новости, литература (журналы, газеты), кино и театральные постановки, интерактивные музеи и пр. (этот список можно продолжать до бесконечности). Последовательное и настойчивое приучение себя к восприятию информации на английском языке помогает быстро преодолеть психологический барьер и начать совершенствоваться в языке.

На наш взгляд, руководствуясь правилом «не пользуйся переводом на русский язык», студенты могут попытаться в какой-то мере моделировать для себя англоязычную среду. Необходимый шаг на пути к достижению этой цели – осознанный выбор англоязычных источников в тех случаях, когда можно воспользоваться переводом на русский язык. В этом случае становится понятно, что речь не идет об использовании компьютера или Интернет-ресурсов, речь идет об использовании языка.

Разумеется, студентов следует побуждать активно искать и использовать эти возможности. Другими словами, они должны сознательно и целенаправленно выстраивать свое собственное англоязычное окружение. Роль и влияние преподавателя, таким образом, не ограничиваются только аудиторными занятиями; его задача более глобальна – в условиях отсутствия англоязычного окружения повысить мотивацию студентов к моделированию «личной» иноязычной среды и активному существованию и самореализации в ее рамках.

Наша концепция создания внеурочной англоязычной среды опирается на некоторые простые практические идеи.

Персонифицированный/личностно-ориентированный/ индивидуальный характер процесса овладения английским языком

Под *индивидуальным характером* процесса овладения иностранным языком мы понимаем следующее: процесс изучения иностранного языка должен быть организован таким образом, чтобы повысить и использовать с максимально возможной отдачей личную заинтересованность студента в постановке и достижении целей. Как правило, при изучении иностранного языка учащиеся вынуждены руководствоваться не личным интересом или своими профессиональными приоритетами, а рекомендациями и требованиями преподавателя-филолога, который может иметь очень поверхностное представление о специальности и специализации своих студентов. Во время аудиторных занятий не все-

гда возможно в полной мере учитывать личные интересы студентов. Однако если говорить о создании внеурочной языковой среды, то ее намного легче и проще «настроить» на тот круг тем, который понятен и интересен студентам или в изучении которых студент был бы по каким-либо причинам в настоящий момент заинтересован.

Поясним эту идею на конкретном примере. В некоторых вузах существует практика внеаудиторного перевода, когда студенты переводят так называемые «тысячи» знаков, используя источники, рекомендованные преподавателем. Как правило, список рекомендованной литературы включает в себя два или три источника (учебники и монографии). Предполагается, что студент самостоятельно выберет главу или статью, содержание которой перекликается с его профессиональными интересами. В действительности *совпадение личных интересов студента и содержания переводимого текста случается крайне редко*.

Во-первых, при выборе материала для перевода необходимо соблюсти требование о недопустимости параллельного/одновременного/независимого перевода одного и того же текста несколькими студентами. Следовательно, если два или три студента в группе интересуются, скажем, вопросами налогообложения, то только один из них будет переводить главу книги, посвященную этому вопросу. Оставшиеся будут вынуждены переводить другие главы с другой тематикой.

Во-вторых, существует вполне реальная возможность того, что из-за ограниченного количества рекомендованных материалов студенты не сумеют выбрать ни одного текста, который был бы им интересен с профессиональной точки зрения и поэтому будут вынуждены пойти на компромисс и довольствоваться переводом текстов, выбранных ими по иным причинам, чаще всего просто наугад.

В-третьих, список рекомендованной литературы не обновляется в течение нескольких лет. В результате содержание материалов, предлагаемых для перевода, успевает потерять свою актуальность и, следовательно, с каждым годом будет представлять все меньший и меньший интерес для студентов.

В итоге, дискредитируется вся идея внеаудиторного перевода: не испытывая никакой личной или профессиональной заинтересованности в усвоении переводимого материала, студенты выполняют перевод крайне небрежно и не совершенствуют свои навыки владения английским языком.

Между тем, выход из сложившейся ситуации видится нам в том, что студенту следует *предоставить возможность подбирать тексты для перевода самостоятельно*, например, англоязычные материалы по

теме курсового проекта, работой над которым он занят в настоящее время. Разумеется, преподаватель оставляет за собой право одобрить или не одобрить тексты, выбранные студентами, но только на основании чисто формальных критериев, таких как количество знаков и т.д. Практика показывает, что в этом случае мотивация студентов значительно повышается и к переводу они подходят более ответственно и осознанно. Более того, их не пугает и возможная повышенная сложность текста – даже в этом случае они предпочитают переводить именно выбранный самостоятельно и необходимый для выполнения курсового проекта текст, зачастую осознанно превышая и установленный преподавателем лимит объема переводимого текста.

Ограничение использования «переводных материалов» и создание учебной англоязычной среды

Наглядным примером в данном случае может выступать Швеция, – страна, где практически все население в большей или меньшей степени владеет английским языком. Англоязычная среда в этой стране создана на уровне государства, можно спорить о достоинствах или недостатках такого агрессивного влияния иного языка, но факт остается фактом, шведский кондуктор трамвая проблем в общении с иностранцами, говорящими на английском языке, не испытывает.

Хотелось бы остановиться на шведском опыте более подробно. Нужно сделать оговорку, речь пойдет о преподавании английского языка в школе (см. об этом [3]). К моменту поступления в университет студенты уже дважды проходят обязательное национальное тестирование (аналог российских ГИА и ЕГЭ). Изучение иностранного языка в университетах не является обязательным. Тем не менее, мы уверены, что поскольку российская практика, к сожалению, показывает, что школьное образование, даже в своей лучшей форме не способно дать ученикам надежное владение английским языком и обеспечить навыки самостоятельного изучения, и университетскому образованию приходится решать эти задачи, этот ценный опыт может и должен быть перенесен в российское образование.

Шведские школы и университеты не используют переводные материалы. Кроме того, изучение английского языка тесно связано с осуществлением обычных школьных видов деятельности и в находится в тесной связи с другими обязательными школьными предметами. Далее, в университете, студенты могут выбрать язык обучения. В последние несколько лет стали появляться исследования по проблемам эффективности получения образования на английском языке. (см. об этом: [2]) В них отмечены в качестве проблемных зон объяснение

сложного материала и использование юмора – традиционные трудности, с которыми сталкивается иностранец, владеющий языком даже на высоком уровне. Тем не менее, данный опыт может быть для российских университетов достаточно ценным. *Создание библиотеки ресурсов на английском языке, поощрение студентов, использующих оригинальные источники,* – все это должно стать обычной практикой, если мы хотим добиться активного использования английского языка в профессиональных целях.

Язык является рабочим инструментом. На наш взгляд, необходимость использования иностранного языка как рабочего инструмента очевидна. Бессмысленно изучать язык ради самого языка (по крайней мере, студентам неязыкового вуза). Студенты должны и могут быть обеспечены плотным профессиональным потоком, дающим возможность совершенствовать свои навыки владения английским языком. Преподаватели, работающие с ресурсами, имеющими перевод на английский язык, могли бы предлагать студентам альтернативу – изучение вопроса на языке оригинала.

Язык является средством коммуникации. Еще одна очевидная идея, часто упускаемая из вида в процессе обучения. Построение урока таким образом, чтобы дать возможность получить навыки общения на английском языке всем студентам в равной мере, является первоочередной задачей преподавателя английского языка.

Формирование учебной автономности. Под формированием учебной автономности мы понимаем формирование потребности саморазвития в целом и навыков изучения иностранного языка с использованием всех доступных ресурсов в частности. Другими словами, важно и необходимо научить студентов учиться, познавать новое, искать и добывать знания. К сожалению, на данный момент ситуация в большинстве вузов такова, что студенты привыкли, что новые знания преподносятся им «на блюдечке» готовыми, загодя, задолго до того, как у них в сознании сформируется запрос на новую информацию. Более того, мотивация на открытие нового, радость от открытия нового, стремление самостоятельно открыть новое отсутствует как факт. Другими словами, студенты не умеют осознавать познавательный интерес и, следовательно, не умеют и не стремятся удовлетворить потребность познания нового. Каким образом идея формирования учебной автономности соотносится с заявленной нами в начале статьи идеей о необходимости формирования языковой среды? На самом деле одно логично вытекает из другого: «автономный студент» с развитыми навыками самостоятельного, осознанного изучения иностранного язы-

ка неизбежно будет искать и создавать возможности выстраивания своего собственного лингвистического окружения.

Заключение

Создание англоязычной среды вне занятий позволяет решить целый ряд задач:

- максимально индивидуализировать процесс изучения иностранного языка и овладения различными видами речевой деятельности на иностранном языке, поскольку приближает его к понятным студентам личным целям и задачам;
- сформировать потребность в самостоятельном развитии;
- «дать в руки» инструменты этого развития.

Поиск эффективных методических решений для проведения качественных занятий, опирающихся на использование средств ИКТ, должен осуществляться с учетом решения этих задач.

Библиографический список

1. *Сысоев П.В., Евстигнеев М.Н.* Использование современных учебных Интернет-ресурсов в обучении иностранному языку и культуре [URL: http://cyberleninka.ru/article/n/ispolzovanie-sovremennyh-uchebnyh-internet-resursov-v-obuchenii-inostrannomu-yazyku-i-kulture]. (Дата обращения: 20.11.2013).
2. *Johansson I., Jonsson B.* Effects of Teaching in English at Swedish Universities // *Mälardalen University*. 2006. Edith Cowan University Research. Available at: http://ro.ecu.edu.au/.
3. *Sundin K.* English as a First Foreign Language for Young Learners – Sweden / An Early Start: Young Learners and Modern Languages in Europe and Beyond. Ed. *M. Nikolov*, *H. Curtain*. Council of Europe, 2000.

S. Atanasov

Trakia University (Yambol, Bulgaria)

s_s_atanasov@abv.bg

AN APPROACH IN TEACHING BASICS OF MICROPROCESSORS

Introduction

The object of this article is processor Motorola 6800. It is still in use over the world (University of Bolton, Kenyatta University etc.) in learning basics of microprocessors, because a small number of instructions and registers, simplified architecture provide very convenient conditions for teaching and understanding the basic principles, operations and processes occurring in microprocessors.

Motorola 6800 is an 8-bit, dual accumulator processor with flexible memory addressing modes. This processor was released at about the same time as Intel 8080. It has six programming accessible registers: two 8-bit accumulators, one 8-bit register for the code of conditions, 16-bit index registers, 16-bit stack pointer and 16-bit program counter. The microprocessor has 72 instructions for arithmetic, logic and control operations. Its clock frequency is 1 MHz. This processor can address 65536 memory cells.

Compared to other processors it's compact. Highly orthogonal instruction set makes it easier to program.

Motorola 6800 hasn't I/O instructions and therefore 6800-based systems have to use memory-mapped I/O for input/output capabilities. Motorola 6800 started the big family of 680x microcontrolers and microprocessors, some of which are still produced today.

The 6800/6811 family of processors fueled the early home computing explosion and its derivatives were the processors of choice for many personal computers including Apple, Commodore64, Nintendo etc, and numerous gaming consoles. Its direct descendants are still widely used today as embedded processors.

Discussion

The main used emulator's name is "Graphical 6800 Simulator" (Fig. 1). The graphical simulator was developed by Mr. S. Scott, an undergraduate student in the Department of Electrical Engineering at the University of Arkansas. Unfortunately it cannot be executed directly on systems newer than Windows XP, because this is 16-bit DOS program. The free tool called DOSbox comes in handy here. We can run this emulator with DOSbox under Windows 7 for example.

Fig. 1. Main screen of the Graphical 6800 Simulator in DOSBox 0.74

A current version of the DOSBox tools is 0.74.

The most interesting possibility of the emulator is that trainees can execute the program on the Assembler step by step. They can observe events occur. Students can see how values in registers and in memory are changed.

The visual environment of the graphical simulator consists of the following panels:

- The panel "MPU Control" oversees the major operations available.
- The panel "Peripheral selection". Any selection produces a depiction of same at lower right of screen. All peripherals are wired to VIA ports A and B.

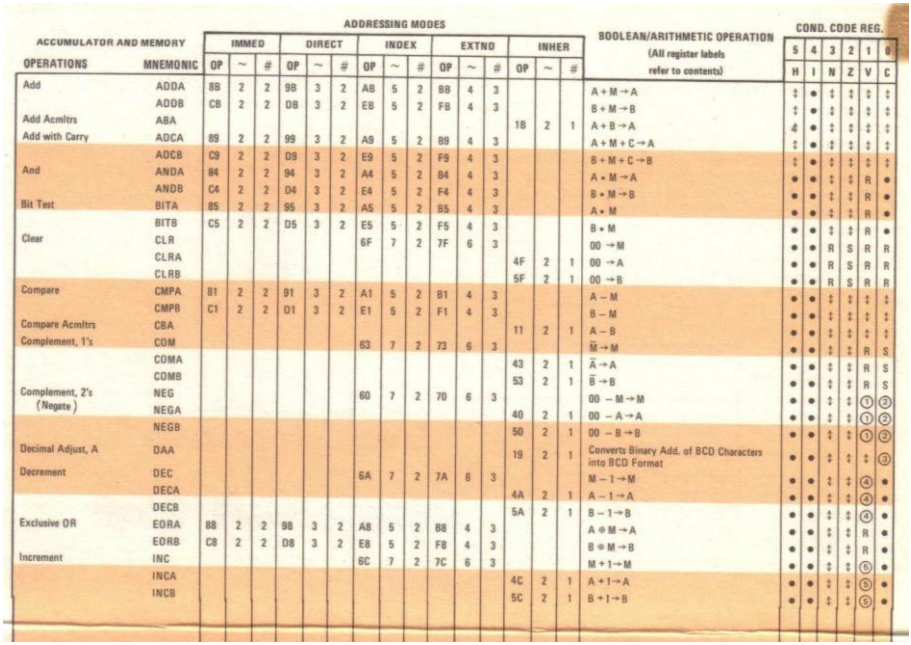
- The panel “Execution Format”. Program can be executed in three modes. For each mode all logic values of data and addresses are highlighted on diagram at lower left of screen, and registers are updated.
- The panel “Run Control”. Program execution can be proceeded or be stopped. Peripheral control allows a key on the hex keypad to be pressed or released, and permits the A/D input voltage to be adjusted.
- The panel “Error messages”. Error messages are posted here.
- Here are noted the current instruction mnemonic, the current clock cycle and total cycles for this instruction (e.g. 1/3) and the total clock cycles, elapsed from the start of the program.

Students receive task, and they have to write assembler code. Good programming style suggests a good comment on the source code. The following program fragment illustrates this:

```
;==============================
; Sum 1 to 10
;==============================
ten         .equ 10
sum10       clra                ;A = 0
            ldab #ten           ;B = 10
loop1       aba                 ;A = A + B
            decb                ;B--
            tstb                ;test (B==0)
            bne  loop1          ;repeat if B!=0
            staa sum            ;sum = A

;==============================
; Sum Array values, zero terminated
;==============================
sumarry     ldx #array          ;X = array
            clra                ;A = 0
            clrb                ;B = 0
loop2       adda 0,x            ;A += array[0]
            inx                 ;next item
            tst  0,x ;check item
            bne  loop2          ;repeat if B!=0
            staa sum            ;sum = A
```

After writing assembler program, students have to translate it manually into hex code, using table below (Fig. 2), which represents machine code, according instruction and type of addressing.



ACCUMULATOR AND MEMORY OPERATIONS	MNEMONIC	ADDRESSING MODES: IMMED OP	IMMED ~	IMMED #	DIRECT OP	DIRECT ~	DIRECT #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	INHER OP	INHER ~	INHER #	BOOLEAN/ARITHMETIC OPERATION (All register labels refer to contents)	COND. CODE REG. 5 H	4 I	3 N	2 Z	1 V	0 C
Add	ADDA	8B	2	2	9B	3	2	AB	5	2	BB	4	3				A + M → A	↕	•	↕	↕	↕	↕
	ADDB	CB	2	2	DB	3	2	EB	5	2	FB	4	3				B + M → B	↕	•	↕	↕	↕	↕
Add Acmltrs	ABA													1B	2	1	A + B → A	↕	•	↕	↕	↕	↕
Add with Carry	ADCA	89	2	2	99	3	2	A9	5	2	B9	4	3				A + M + C → A	↕	•	↕	↕	↕	↕
	ADCB	C9	2	2	D9	3	2	E9	5	2	F9	4	3				B + M + C → B	↕	•	↕	↕	↕	↕
And	ANDA	84	2	2	94	3	2	A4	5	2	B4	4	3				A • M → A	•	•	↕	↕	R	•
	ANDB	C4	2	2	D4	3	2	E4	5	2	F4	4	3				B • M → B	•	•	↕	↕	R	•
Bit Test	BITA	85	2	2	95	3	2	A5	5	2	B5	4	3				A • M	•	•	↕	↕	R	•
	BITB	C5	2	2	D5	3	2	E5	5	2	F5	4	3				B • M	•	•	↕	↕	R	•
Clear	CLR							6F	7	2	7F	6	3				00 → M	•	•	R	S	R	R
	CLRA													4F	2	1	00 → A	•	•	R	S	R	R
	CLRB													5F	2	1	00 → B	•	•	R	S	R	R
Compare	CMPA	81	2	2	91	3	2	A1	5	2	B1	4	3				A − M	•	•	↕	↕	↕	↕
	CMPB	C1	2	2	D1	3	2	E1	5	2	F1	4	3				B − M	•	•	↕	↕	↕	↕
Compare Acmltrs	CBA													11	2	1	A − B	•	•	↕	↕	↕	↕
Complement, 1's	COM							63	7	2	73	6	3				M → M	•	•	↕	↕	R	S
	COMA													43	2	1	Ā → A	•	•	↕	↕	R	S
	COMB													53	2	1	B̄ → B	•	•	↕	↕	R	S
Complement, 2's (Negate)	NEG							60	7	2	70	6	3				00 − M → M	•	•	↕	↕	①	②
	NEGA													40	2	1	00 − A → A	•	•	↕	↕	①	②
	NEGB													50	2	1	00 − B → B	•	•	↕	↕	①	②
Decimal Adjust, A	DAA													19	2	1	Converts Binary Add. of BCD Characters into BCD Format	•	•	↕	↕	↕	③
Decrement	DEC							6A	7	2	7A	6	3				M − 1 → M	•	•	↕	↕	④	•
	DECA													4A	2	1	A − 1 → A	•	•	↕	↕	④	•
	DECB													5A	2	1	B − 1 → B	•	•	↕	↕	④	•
Exclusive OR	EORA	88	2	2	98	3	2	A8	5	2	B8	4	3				A ⊕ M → A	•	•	↕	↕	R	•
	EORB	C8	2	2	D8	3	2	E8	5	2	F8	4	3				B ⊕ M → B	•	•	↕	↕	R	•
Increment	INC							6C	7	2	7C	6	3				M + 1 → M	•	•	↕	↕	⑤	•
	INCA													4C	2	1	A + 1 → A	•	•	↕	↕	⑤	•
	INCB													5C	2	1	B + 1 → B	•	•	↕	↕	⑤	•

Fig. 2. Part of the table with HEX codes of mnemonic instructions

To strengthen the individual work of students the following table must be filled and entered manually on the screen (Fig. 3):

Mnemonic code			Type of used addressing	Hex			B	Memory address	Comment
Label	Instr.	Operand							
	LDAA	$40	direct	96	40		2	0000	;M->A
	ANDA	#%0000111	immediate	84	0F		2	0002	;A^M->A
	STAA	$41	direct	97	42		2	0004	;A->M
	LDAA	$40	direct	96	40		2	0006	;M->A
	LSRA		internal	44			1	0008	;0->b7
	LSRA		internal	44			1	0009	;0->b7
	LSRA		internal	44			1	000A	;0->b7
	LSRA		internal	44			1	000B	;0->b7
	STAA	$42	direct	97	41		2	000C	;A->M
	SWI		internal	3F			1	000E	

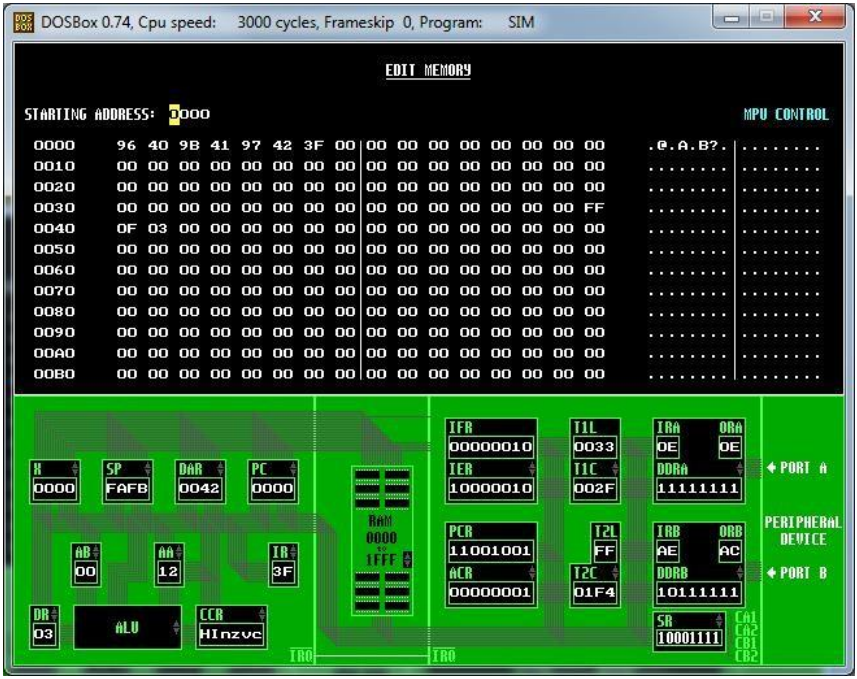


Fig. 3. “Edit Memory” screen

The “Edit Memory” selection produces a view of 128 bytes of memory beginning at the user-specified starting address. Both the hex value and its ASCII translation are displayed. Any visible location can be edited. An exit back to the main panel can be executed.

In addition there is other free tool called “SDK6800 Emulator”. Its advantage is that inputted assembly code, if it is written correctly, appears on the left side of the screen like machine code, and there is no need to enter code manually. The program also includes tools for tracking registers and executing processes step by step (Fig. 4).

“6800IDE” is a freeware IDE for Motorola's 6800/6811 processor. This IDE is based on Windows. Designed for educational purposes, it includes an assembler and an emulator for the 6800/6811 processor with built-in debugging tools support such as user breakpoints, execution trace, internal register display and a Hex/Bin/Dec number convertor.

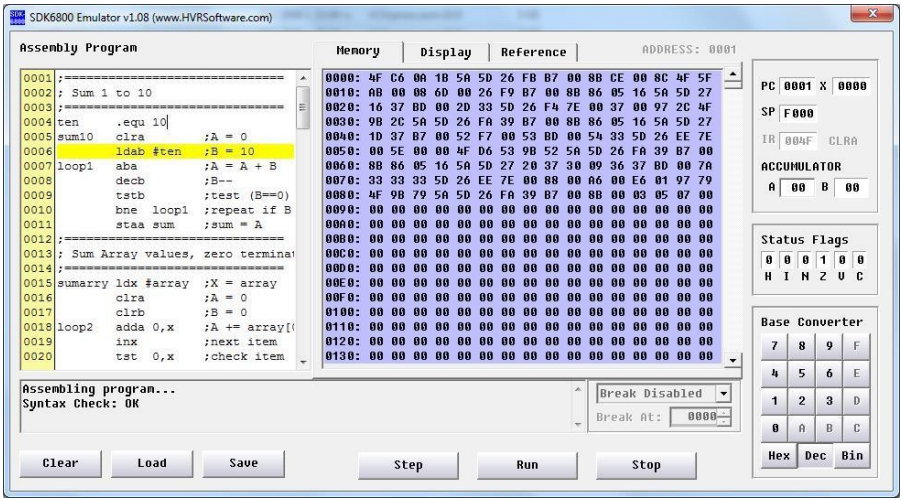


Fig. 4. SDK6800 Emulator v.1.08

Conclusion

We decided to use combination of two tools, thus the combination gave us the following advantages: the above mentioned emulators and practices demonstrate to the students how the MPU behaves during the individual clock cycles necessary to complete the instruction, particularly because it is in graphical form. We find that students have some difficulty in understanding complex assembly programs, and again the graphical display of operation eases this obstacle. Best of all, the simulator is easy to learn; no intensive study of a thick manual is required. The students were able to quickly operate all aspects and then spend their time on the device details rather than that of the software.

References

1. *Haggard R.L.* Classroom Experiences and Student Attitudes toward Electronic Design Automation // Proceedings of CA, IEEE Computer Society Press, 1993. P. 411-415.
2. *Leventhal L.* 6800 Assembly Language Programming, 1978 [Available at: http://www.hvrsoftware.com/6800emu.htm].

СОДЕРЖАНИЕ

Научное издание

Математика программных систем

Межвузовский сборник научных трудов

Выпуск 10

Редактор *Е.В. Шумилова*

Компьютерная вёрстка *Л.Н. Лядовой*

Подписано в печать 25.11.2013. Формат 60×84 $^{1}/_{16}$.
Усл. печ. л.9,88. Тираж 100 экз.

Издательский центр
Пермского государственного
национального исследовательского университета.
614990, г. Пермь, ул. Букирева, 15

Отпечатано в ООО «Учебный центр “Информатика”»
614990, г. Пермь, ул. Букирева, 15