ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г.

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана, ОАО "Концерн "Сириус".

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индексы: по каталогу агентства "Роспечать" — **22765**, по Объединенному каталогу "Пресса России" — **39795**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru/pi.html E-mail: prin@novtex.ru

Журнал включен в систему Российского индекса научного цитирования.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

№ 9
September
2014

Published since September 2010

CONTENTS

**Information about the jornal is available online at:
http://novtex.ru/pi.html, e-mail:prin@novtex.ru**

УДК 004.4

В. А. Васенин, д-р физ.-мат. наук, проф., зав. лаб., e-mail: vasenin@msu.ru,
Д. Д. Голомазов, канд. физ.-мат. наук, ст. науч. сотр., e-mail: denis.golomazov@gmail.com,
Г. М. Ганкин, программист, gregorian21@yandex.ru, НИИ механики МГУ
им. М.В. Ломоносова

Архитектура, методы и средства базовой составляющей системы управления научной информацией "ИСТИНА – Наука МГУ"

Приведено описание общей архитектуры и функций базовой составляющей Интеллектуальной системы тематического исследования научно-технической информации (ИСТИНА). Система разработана в МГУ имени М. В. Ломоносова. Ее целью является сбор, учет, систематизация, хранение, анализ и выдача по запросу информации, характеризующей результаты деятельности научных и образовательных организаций. Представлены общая архитектура системы ИСТИНА и принципы работы ее компонентов (модулей). Особое внимание уделено методам и средствам, применяющимся в ее базовых программных модулях, включая механизмы добавления новых типов результатов деятельности и средства редактирования данных. Содержание статьи в основном имеет практический характер. В относительно общем виде в ней обсуждаются вопросы программно-технического характера, возникающие на различных этапах жизненного цикла системы.

Ключевые слова: *научная информация, информационно-аналитическая система, научная деятельность, оценка эффективности, библиографическая система, метаданные*

Введение

Интеллектуальная система тематического исследования научно-технической информации (ИСТИНА) [1] предназначена для управления информацией, характеризующей результаты деятельности научных и образовательных организаций. Основные функции системы, более подробно описанные в работе [2], включают ввод данных, отображение их в открытом доступе, формирование отчетов и представление данных для сотрудников в различных форматах, а также анализ статистических показателей по подразделениям и по тематикам. Отличительными особенностями системы являются: широкий охват результатов научной деятельности; контроль пользователей над данными; удобство ввода информации; использование онтологий для учета специфики различных областей знаний; полезные функции для сотрудников; широкие возможности анализа для руководителей; масштабируемость.

Одной из главных целей статьи является желание показать, за счет каких методов и программных механизмов реализуются функции системы, которые отличают ее от других активно используемых в научном сообществе систем подобного рода, например, Web of Science и Scopus. С самых общих позиций такое отличие, во-первых, связано с тем, что основное внимание уделяется архитектуре системы и структуре кода, которые обеспечивают ее расширяемость в сторону поддержки новых видов результатов деятельности. Во-вторых, значительное внимание уделяется функциям, позволяющим пользователям и ответственным сотрудникам отдельных подразделений организаций самим добавлять, редактировать и исправлять некорректные данные.

Первая часть статьи содержит общую информацию о системе, ее основных функциях, об отличительных особенностях, архитектуре и основных технологиях, которые в ней используются. Архитектура системы включает такие компоненты, как сервер базы данных и сервер приложений. Система построена с применением таких языков программирования и технологий, как Python, Django, Oracle, Nginx, а также ряда других. Представлены основные преимущества веб-фреймворка Django и его отличия от других подобных библиотек. Кратко описана схема обработки HTTP-запроса в Django, включающая обработку адреса (URL) запроса, вызов соответствующей Python-функции в модуле View, обращение к модели и затем к базе данных с помощью объектно-реляционного отображения (ORM), формирование ответа на языке HTML с помощью заполнения шаблона. Далее представлена структура кода Django-приложений, составляющих большую часть системы. Основу системы составляет ее базовая часть — ядро, на которое опираются приложения "организации" и "сотрудники", приложения класса "результаты деятельности", приложения "статистика" и "онтологии", а также сторонние приложения, реализующие вспомогательные функции.

Во второй части статьи описаны основные составляющие ядра системы: модели, формы и "мастера форм" (*wizards*). Представлены их структура и способы исполь-

зования. На основе ядра реализован механизм, при помощи которого система унифицированно обрабатывает различные типы деятельности сотрудников. Он включает в себя генерацию интерфейса, позволяющего пользователям добавлять, редактировать, просматривать и удалять работы заданного типа деятельности. Генерация интерфейса выполняется с помощью "мастеров форм", которые являются частью фреймворка Django. Упомянутый механизм позволяет добавлять в систему поддержку новых видов результатов деятельности, минимально изменяя имеющийся код. Кроме того, описаны другие функции ядра, которые можно разделить на две группы. В первую группу входят функции, целью которых является очистка данных на этапе ввода в систему (поиск похожих сотрудников) или после него (слияние похожих работ). Ко второй группе относятся служебные функции, реализующие механизмы кэширования, журналирования, версионности, контроля ошибок и др.

1. Архитектура системы и технологии

Взаимодействие пользователей с системой ИСТИНА осуществляется через веб-интерфейс, архитектура системы является типичной для современных веб-приложений (рис. 1).

В основе системы лежит сервер приложений. Главным языковым средством для разработки приложений в системе в настоящее время является веб-фреймворк Django[1], который написан на языке Python. Для изоляции пакетов на сервере приложений и предотвращения конфликтов их версий используют модуль виртуального окружения Virtualenv[2] и вспомогательный модуль Virtualenvwrapper[3]. В качестве веб-сервера применяется HTTP-сервер Nginx[4]. Взаимодействие между веб-сервером и Python-кодом приложений осуществляется по стандарту WSGI[5] с помощью программы uWSGI[6]. В качестве СУБД используется Oracle, физически расположенный на отдельной машине. Для ускорения доступа к данным применяется система управления кэшем Memcached[7]. Для контроля состояния основных процессов на сервере приложений используется программа Supervisor[8]. В случае аварийного завершения какого-либо из процессов он автоматически перезапускается. Ответ пользователю, сгенерированный на сервере приложений с использованием информации из базы данных, в общем случае является HTML-документом. Для управления отображением элементов в HTML-документах применяется CSS-фреймворк Blueprint CSS[9], а для задания поведения элементов на стороне клиента — Javascript-библиотека jQuery[10].

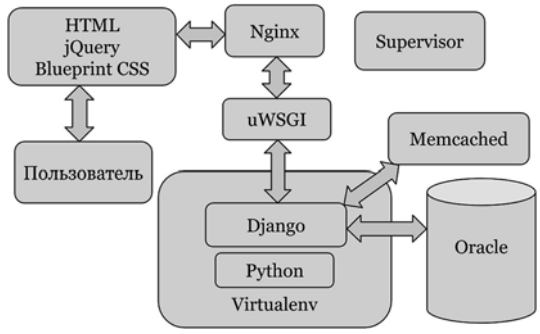


Рис. 1. Архитектура системы ИСТИНА

Как было отмечено выше, основной код системы написан с использованием веб-фреймворка Django. Этот фреймворк реализует вариант шаблона проектирования "модель—представление—поведение" (Model—View—Controller, MVC[11]), включает средства объектно-реляционного отображения (ORM), позволяющие автоматически генерировать SQL-запросы на основе Python-кода, а также включает встроенный язык шаблонов и поддерживает независимое задание URL-адресов для веб-сайта. Отметим, что эти функции в том или ином виде входят в большинство современных веб-фреймворков. Необходимо заметить, что шаблон проектирования MVC позволяет разделить на отдельные компоненты модель данных приложения, пользовательский интерфейс (представление) и связь между пользователем и системой (поведение/контроллер). Такое разделение позволяет повысить возможность повторного использования кода и облегчить его модификацию. Выбор именно Django кроме исторических причин обусловлен рядом достоинств этого фреймворка, которые выделяли его на фоне других библиотек на момент принятия решения о создании системы в 2011 г. Во-первых, привязка к языку Python позволяет использовать богатый набор других библиотек языка, которые реализуют всевозможные функции, причем не только связанные с веб-интерфейсом системы. Во-вторых, в Django изначально заложена экосистема повторно используемых приложений (*third-party apps*), реализующих функциональные возможности, разделяемые различными веб-сайтами. Например, в системе ИСТИНА применяется стороннее Django-приложение для регистрации пользователей. Более подробный список используемых приложений представлен в следующих разделах. В-третьих, в Django на основе заданных программистом моделей автоматически генерируется полноценный веб-интерфейс администратора, с помощью которого он может просматривать, добавлять, изменять и удалять всю необходимую информацию в базе данных. Наконец, эффективность и масштабируемость Django подтверждается тем фактом, что библиотека используется в ряде крупнейших веб-проектов с десятками и даже сотнями миллионов пользователей, таких как Instagram[12],

[1] https://www.djangoproject.com/
[2] http://www.virtualenv.org
[3] http://virtualenvwrapper.readthedocs.org
[4] http://nginx.org
[5] http://legacy.python.org/dev/peps/pep-3333
[6] http://uwsgi-docs.readthedocs.org
[7] http://memcached.org
[8] http://supervisord.org
[9] http://www.blueprintcss.org
[10] http://jquery.com

[11] http://heim.ifi.uio.no/trygver/themes/mvc/mvc-index.html
[12] http://instagram.com

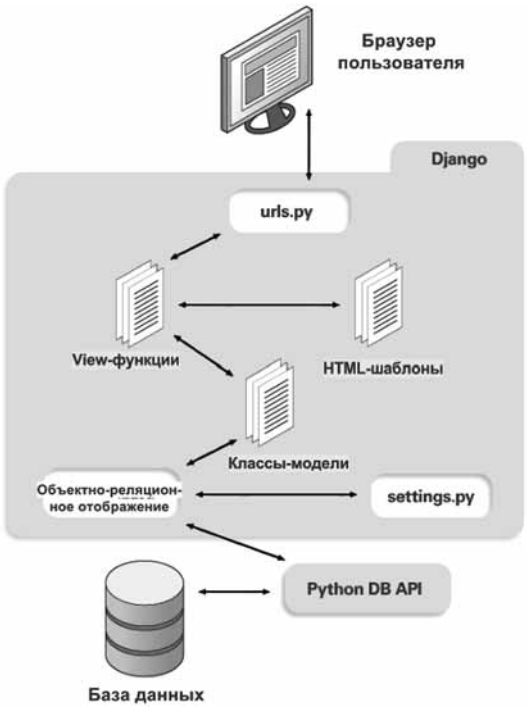


Рис. 2. Схема обработки HTTP-запроса фреймворком Django

Pinterest[13], Disqus[14], Mozilla[15], Bitbucket[16], Mahalo[17], Rdio[18], Lanyrd[19], The Onion[20], на веб-сайтах крупнейших изданий, таких как The Washington Post[21], The New York Times[22], The Guardian[23].

На рис. 2 представлена общая схема обработки HTTP-запроса фреймворком Django. Запрос пользователя, первоначально обработанный веб-сервером, попадает через программу-коннектор (в данном случае – uWSGI) в модуль urls.py, в котором происходит сравнение URL-запроса со списком регулярных выражений, определенном в приложении. Каждому регулярному выражению соответствует Python-функция, называемая view-функцией, которая исполняется в случае, когда адрес запроса подходит под это выражение. Отметим, что исполняется только одна функция, которая соответствует первому в списке выражению, под которое подошел запрос. View-функции находятся в файлах views.py внутри приложений. Эти функции принимают на вход объект request, содержащий данные HTTP-запроса, а также параметры запроса из его адресной строки. Во view-функции может происходить обращение к методам сущностей-моделей, определенных в виде классов в файле models.py. Во view-функциях, а также в моделях выполняются запросы к базе данных с помощью ORM-отображения. В файле settings.py находятся общие параметры проекта, включающие настройки базы данных. Во view-функции формируются данные для отображения пользователю. Эти данные передаются в заданный шаблон (написанный, например, на языке HTML) и далее заполненный HTML-документ отправляется через программу-коннектор и веб-сервер обратно пользователю.

[13] https://www.pinterest.com
[14] http://disqus.com
[15] http://www.mozilla.org
[16] https://bitbucket.org
[17] http://www.mahalo.com
[18] http://www.rdio.com
[19] http://lanyrd.com
[20] http://www.theonion.com
[21] http://www.washingtonpost.com
[22] http://www.nytimes.com
[23] http://www.theguardian.com

2. Структура кода приложений

Общая структура кода приложений системы представлена на рис. 3. Основа системы — базовая составляющая — **ядро**, реализующее системообразующие функции, которые во многом определяют показатели качества системы в целом. Оно содержит базовые классы, методы, формы и функции, используемые в других модулях системы. Ядро обеспечивает единый интерфейс и базовый каркас добавления, просмотра, редактирования и удаления результатов научно-педагогической деятельности сотрудников. В ядре содержатся функции поиска похожих объектов, которые используются для подбора, в частности, похожих сотрудников, журналов и статей. Важной частью ядра является механизм кэширования. Он используется для ускорения доступа к различным данным системы. В ядре реализуются общие функции, поддерживающие принятую политику обеспечения информационной безопасности системы, в первую очередь, механизмы разграничения доступа к информации о результатах деятельности сотрудников.

На следующем уровне структуры кода расположены два базовых приложения — **модуль "Организации"** и **модуль "Сотрудники"**. Отметим, что в настоящем разделе под приложением понимается логически и физически отдельный программный модуль системы, реализующий ту или иную базовую или прикладную функцию. При этом каждое приложение одновременно является и приложением в терминологии веб-фреймворка Django (*Django app*). Модуль "Организации" включает классы и логику действий, связанную с обработкой данных на уровне организаций. В частности, этот модуль включает механизмы, реализующие действия сотрудников, которым делегирована роль ответственных от организации (в данном случае — МГУ) и отдельных ее подразделений. Он обеспечивает добавление, просмотр, редактирование и удаление информации об ответственных лицах, которые призваны решать вопросы заданного пользователя в зависимости от его места работы. Приложение "Сотрудники" содержит базовые классы, связанные с сотрудниками, в частности, их профиль в системе и список работ. Эти два модуля являются базовыми приложениями потому, что с ними связаны все остальные приложения системы, которые находятся выше по структурной иерархии.

Основной уровень структуры кода составляют приложения класса **"Результаты деятельности"**. Каж-

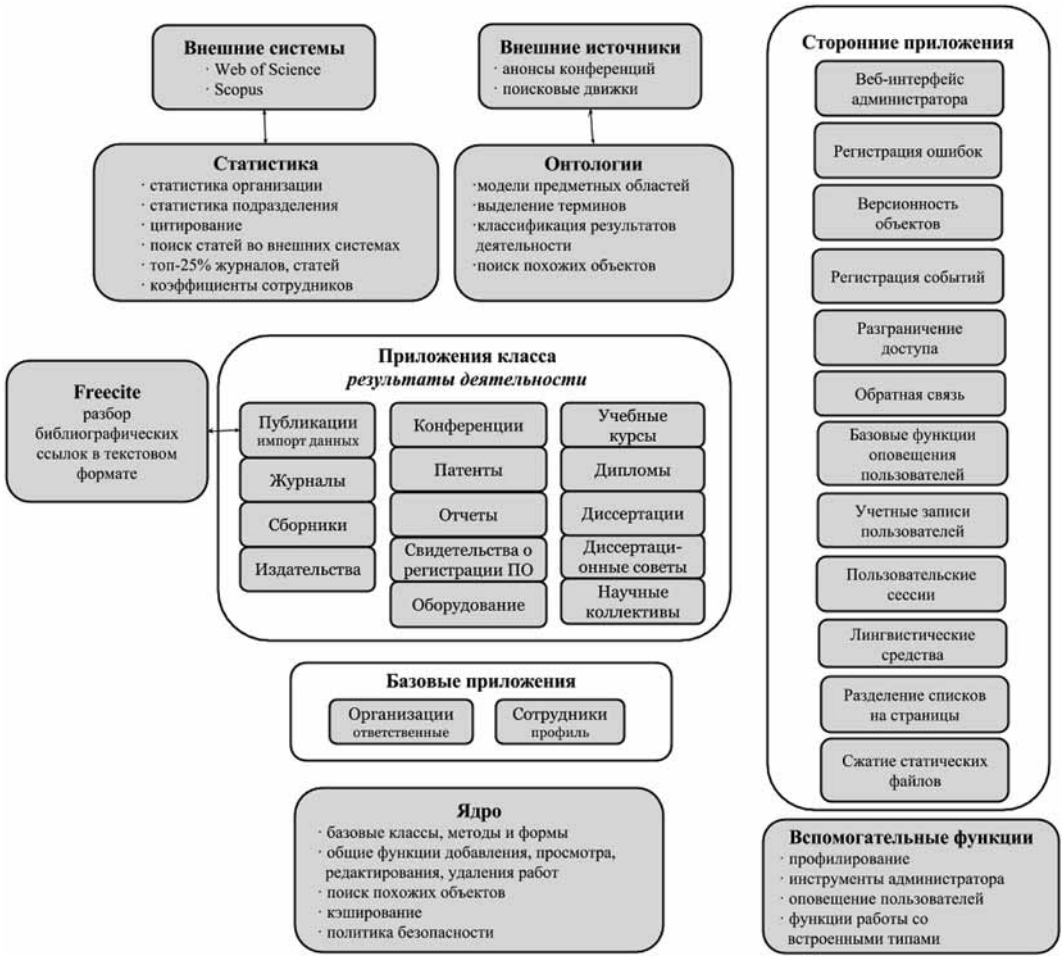


Рис. 3. Структура кода приложений системы ИСТИНА

дое такое приложение отвечает за отдельный тип результатов научно-инновационной и преподавательской деятельности сотрудников (например, публикации) или за общую сущность, связанную с таким типом (например, журналы). Эти приложения опираются на общий каркас классов и функций системы, которые заложены в ее ядре. Такой подход позволяет сравнительно легко добавлять новые типы результатов деятельности персоналиев в систему. Такие функциональные возможности, как добавление, просмотр, редактирование и удаление, а также привязка результата деятельности к сотрудникам-авторам уже реализованы в ядре. Программисту необходимо лишь настроить каркас на конкретный тип результатов деятельности, указав специфицирующие его свойства. Отметим, что наиболее сложным приложением из рассматриваемого класса является модуль "Публикации". Он использует модуль разбора библиографических ссылок в текстовом формате Freecite[24], который был усовершенствован для возможности обработки текстов на русском языке.

На следующем уровне иерархии в структуре кода системы расположены два модуля, отвечающие за анализ данных, накопленных на предыдущих уровнях. **Модуль "Статистика"** содержит функции преимущественно количественного анализа данных системы. Он включает в себя функции анализа данных на уровне организации, подразделения и отдельного сотрудника. В этом модуле реализован также тематический анализ показателей результативности сотрудников, который выполнен сравнительно простыми средствами, а именно: статья в журнале считается принадлежащей тематической рубрике, к которой принадлежит сам журнал. Приложение "Статистика" включает функции получения и обработки показателей цитирования отдельных статей из системы Web of Science и Scopus, а также поиск статей в этих системах. Кроме этого, в модуль входят механизмы расчета принятых в организации коэффициентов эффективности деятельности сотрудников на основе списка их статей в высокорейтинговых журналах. Эти коэффициенты в дальнейшем могут использоваться при расчете поощрительных надбавок сотрудникам.

Приложение "Онтологии" имеет целью выполнение более глубокого тематического анализа данных в системе. В частности, механизмы этого модуля основаны на моделях предметных областей, построенных

[24] http://freecite.library.brown.edu/

в автоматизированном режиме, для этого используются анонсы научных конференций, а также результаты запросов к поисковым системам, таким как Bing[25]. Эти модели содержат термины, характерные для соответствующей предметной области. По этим терминам затем выполняется классификация результатов деятельности сотрудников (например, статей), а также подбор похожих объектов для удобной навигации по системе. При решении данных задач используют различные методы обработки и анализа текстов на естественном языке, например, построение иерархии понятий на основе лексических шаблонов [3], выделение терминов из коллекции текстов [4]. Отметим, что значительная часть функций этого модуля в настоящее время находится в стадии разработки.

Кроме основной вертикали приложений, которая представлена в левой части рис. 3, в системе используются следующие вспомогательные приложения и функции:

- профилирование, т. е. оценка производительности и других характеристик системы (в частности, оценка скорости загрузки краткой информации о сотруднике и списка последних добавленных в систему работ);
- инструментальные средства администратора для использования в командной строке и в веб-интерфейсе системы;
- функции оповещения пользователей о различных событиях в системе;
- вспомогательные функции работы со встроенными типами языка Python (такими как строки, кортежи, списки и словари);
- веб-интерфейс администратора, предоставляемый фреймворком Django;
- подсистема регистрации ошибок на сайте (приложение django-sentry[26]);
- подсистема регистрации и хранения версий объектов, используемая для обеспечения безопасности данных системы, а также для ведения дополнительного журнала событий на уровне всей системы и отдельного пользователя (приложение django-reversion[27]);
- подсистема регистрации и отображения событий различного уровня в системе (приложение django-activity-stream[28]);
- механизмы разграничения доступа к отдельным объектам в системе (приложение django-object-permission[29]);
- функции обратной связи с разработчиками системы и регистрации сообщений пользователей;
- базовые функции оповещения пользователей (приложение django.contrib.messages, входящее в состав Django);
- приложение для работы с учетными записями пользователей (приложения django-profile[30] и django.contrib.auth);
- приложение для работы с пользовательскими сессиями (django.contrib.sessions);
- средства для работы с русским языком (например, склонение числительных — приложение pytils[31]);
- функции разделения текста на страницы (пагинации) для удобного отображения (приложение django-paging[32]);
- приложение для сжатия и версионности статических файлов (django-compressor[33]).

3. Базовая составляющая

Базовая составляющая системы, именуемая ядром, реализует классы, методы, формы и функции, общие для многих типов объектов, с которыми система имеет дело. Ядро обеспечивает следующие функциональные возможности:

- единый интерфейс добавления, просмотра, редактирования и удаления результатов деятельности сотрудников;
- поиск похожих объектов, например, сотрудников, журналов и статей;
- слияние объектов, например, сотрудников или журналов;
- кэширование;
- общие функции, реализующие политику безопасности системы, в частности, механизмы разграничения доступа к результатам деятельности сотрудников;
- вспомогательные функции.

Далее эти функции описаны подробнее.

3.1. Общий код для работы с результатами деятельности

Важнейшим отличием системы ИСТИНА от других похожих систем является ее изначальная ориентация не только на статьи в журналах, на иные публикации, но и на многие другие виды результатов научной, научно-педагогической и научно-административной деятельности. Механизмы для поддержки многих видов подобных сущностей заложены в ядре системы. Эти механизмы позволяют автоматически получать работающий интерфейс добавления, просмотра, редактирования и удаления нового типа результатов деятельности на основе его описания в виде соответствующей модели. Далее более подробно описано, как реализован этот подход, который включает следующие части:

- иерархия базовых классов-моделей, от которых наследуются классы, представляющие отдельные виды результатов деятельности;
- иерархия базовых форм для ввода данных пользователем;
- иерархия мастеров форм (*wizards*), которые обеспечивают логику и интерфейс добавления и редактирования объектов;
- функции поиска похожих объектов;
- функции слияния объектов;
- другие вспомогательные функции.

[25] https://www.bing.com
[26] https://github.com/getsentry/sentry
[27] https://github.com/etianen/django-reversion
[28] https://github.com/justquick/django-activity-stream
[29] https://pypi.python.org/pypi/django-object-permission/
[30] https://code.google.com/p/django-profile/
[31] https://github.com/j2a/pytils
[32] https://github.com/dcramer/django-paging
[33] https://github.com/jezdez/django_compressor

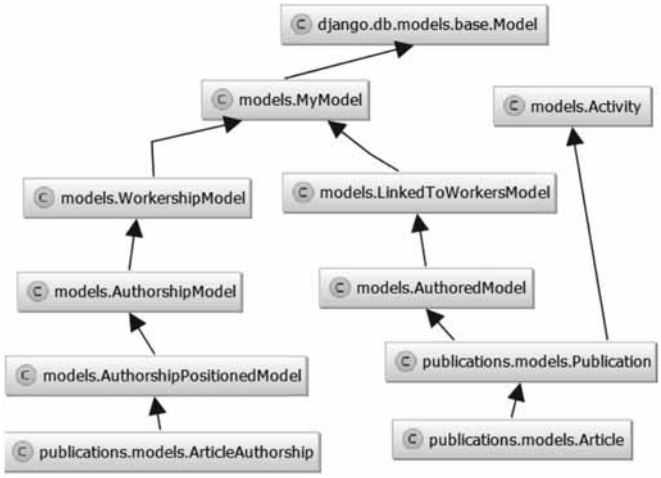


Рис. 4. Иерархия базовых классов-моделей в системе ИСТИНА (фрагмент)

В настоящем подразделе представлено описание первых трех частей, а именно базовых классов, форм и мастеров форм. В классах-моделях описываются общие атрибуты объектов этих классов. Основная часть иерархии базовых классов-моделей в системе ИСТИНА представлена на рис. 4. Базовым классом, который определен в системе и наследуется от Django-класса Model, является класс MyModel. В нем определены разделяемые другими классами самые общие атрибуты и методы. Далее иерархия классов разделяется на две цепочки. Первая из них, которая начинается с класса WorkershipModel, служит для задания объектов — результатов деятельности, связанных только с одним сотрудником, например, "авторство статьи", "авторство патента", "руководство диссертацией", "членство в научном обществе", "членство в редколлегии журнала". Вторая цепочка начинается с класса LinkedToWorkersModel и служит для задания объектов — результатов деятельности, которые могут быть связаны с группой сотрудников, например, статья, патент, отчет, НИР, доклад на конференции, учебный курс. Объекты этих типов связываются с классом Worker (сотрудник) именно через объекты классов, наследованных от WorkershipModel, например, статья связана с сотрудниками-авторами через объекты класса "авторство статьи". В классах WorkershipModel и LinkedToWorkersModel заданы основные методы, относящиеся к связи "сотрудник—результат деятельности", например: получение списка сотрудников по объекту — результату деятельности в различных видах; добавление сотрудников к объекту; изменение связи между объектом и сотрудником.

На следующем уровне иерархии классов находятся модели AuthorshipModel и AuthoredModel. Они служат для задания результатов деятельности, у которых роль связанных сотрудников называется "автор". Примерами могут служить авторства статей (наследуется от AuthorshipModel)/статьи (наследуется от AuthoredModel), авторства патентов/патенты, авторства учебных курсов/учебные курсы. Эти две модели созданы для уменьшения повторного кода и не включают сложнореализуемых функций. От класса AuthorshipModel наследуется класс AuthorshipPositionedModel, отличающийся от базового класса тем, что служит для задания результатов деятельности, для которых важен порядок авторов, например, для статей и патентов. Отдельным классом, наследующимся напрямую от MyModel, является модель Activity. Этот класс не содержит никаких функций и служит своеобразной меткой для классов, задающих отдельные виды результатов деятельности. Если такой класс наследуется, в том числе, и от Activity, то это означает, что он является "окончательным" классом, обозначающим результат деятельности. Именно эти типы объектов перечислены в качестве вариантов на странице добавления нового результата деятельности. Кроме этого, факт "окончательности" используется для формирования списка типов объектов на странице сотрудника, а также в общей статистике системы. Отметим, что "окончательными" классами могут быть как подклассы WorkershipModel (например, членства в научных обществах), так и подклассы LinkedToWorkersModel (например, статьи, патенты или диссертации).

На нижнем уровне иерархии находятся классы, задающие отдельные виды результатов деятельности сотрудников, например, авторство статьи (ArticleAuthorship), отчет, диссертация. Отметим, что для публикаций создан промежуточный класс Publication, в котором содержится общая для двух классов функция публикации в системе, статей (в журналах и сборниках) и книг.

Основная часть иерархии базовых форм в системе ИСТИНА представлена на рис. 5. На верхнем уровне иерархии находится класс MyBaseForm, в котором задаются общие css-классы для отображения полей определенных типов во всех формах. Например, если в какой-либо форме на сайте поле называется date, startdate или enddate, то для него при отображении полностью автоматически генерируется всплывающий календарь для удобного выбора даты. Отметим, что при необходимости это поведение (как и другое, заданное в базовых классах) можно отключить для отдельной формы. Далее следует класс MyModelForm, в котором определяются общие методы, вызываемые при

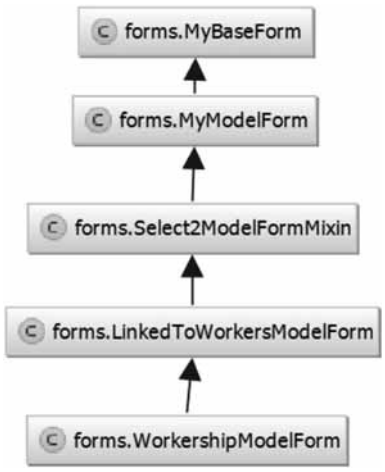


Рис. 5.Иерархия базовых форм в системе ИСТИНА (фрагмент)

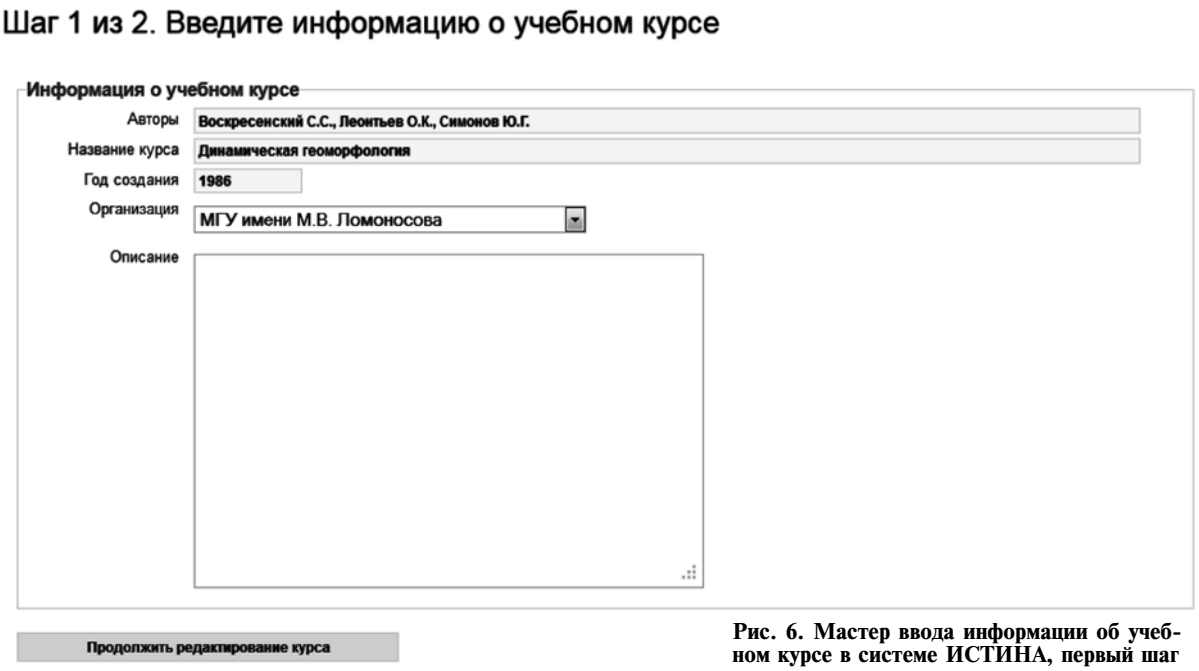


Рис. 6. Мастер ввода информации об учебном курсе в системе ИСТИНА, первый шаг

сохранении объектов по заполненным в форме данным например, запись создателя объекта. В следующем, вспомогательном классе Select2ModelFormMixin определяются функции, связанные с интерактивным интерфейсом выбора сотрудников для связи с результатом деятельности. Он основан на стороннем модуле django-select2[34]. Наконец, в классах LinkedToWorkersModelForm и WorkershipModelForm задаются методы сохранения связи между результатами деятельности (объектами-потомками классов LinkedToWorkersModel и WorkershipModel соответственно) и выбранными пользователем сотрудниками, а также поля для автодополнения имен сотрудников в этих формах.

Основная часть общей логики, связанной с добавлением и редактированием результатов деятельности, находится в классах, задающих мастера форм. Мастера форм (*wizards* в терминологии Django) — это классы, облегчающие реализацию форм на веб-сайте, состоящих из нескольких шагов. Пример отдельного шага мастера форм представлен на рис. 6. Мастера форм позволяют сохранять какую-либо информацию между шагами, динамически включать или исключать шаги на основе введенных пользователем данных, а также сохранять всю введенную информацию после того, как все необходимые формы были заполнены и проверены. В системе ИСТИНА определены два основных класса мастеров форм (рис. 7): MyModelWizardView и LinkedToWorkersModelWizardView. В первом из них реализованы общие функциональные возможности, например, проверка прав доступа; механизмы динамического включения/исключения шагов; сохранение и получение общих данных между шагами. Во втором классе задана логика, отвечающая за шаг разрешения неоднозначностей имен сотрудников. На этом шаге по фамилии и инициалам, введенным пользователем (например, Иванов А. А.) подбираются похожие сотрудники из базы данных системы и отображается краткая информация о них, чтобы пользователь мог выбрать нужного сотрудника из однофамильцев. Система также отображает сотрудников с фамилиями, отличающимися на одну или две буквы, учитывая тот факт, что пользователь мог ошибиться при вводе фамилии. Во многих случаях система может с высокой степенью уверенности сделать выбор самостоятельно, при этом один из сотрудников отображается пользователю как выбранный по умолчанию. Пользователь может при необходимости скорректировать выбор.

На настоящее время алгоритм разрешения неоднозначностей основан на использовании расстояния Левенштейна для определения схожести фамилий. Пример разрешения неоднозначностей фамилий представлен на рис. 8. В дальнейшем планируется применять алгоритм с использованием машинного обучения [5].

В классе LinkedToWorkersModelWizardView также определены методы, сохраняющие выбранные связи между введенными фамилиями и инициалами, со-

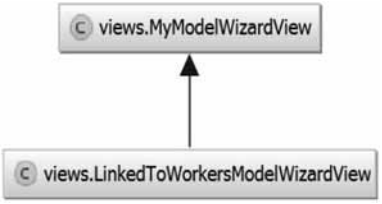
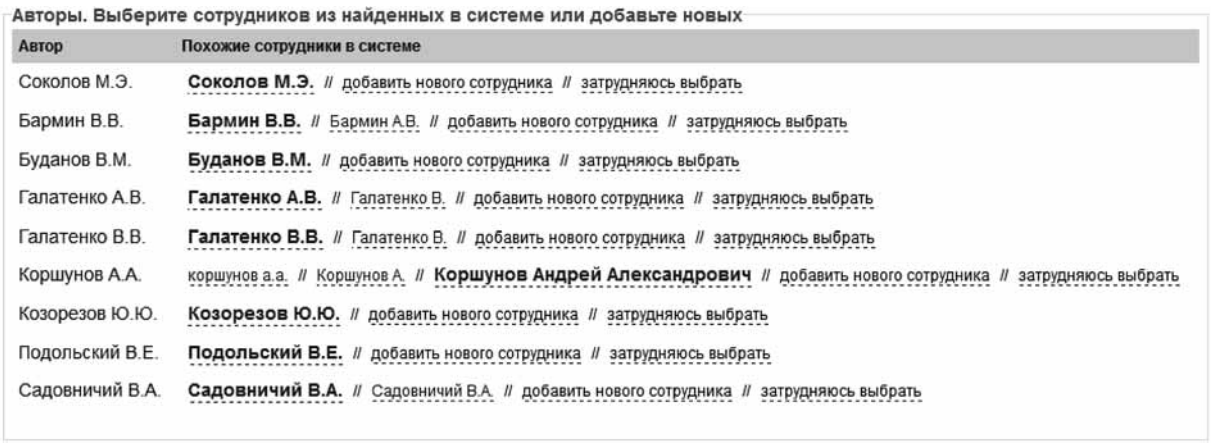


Рис. 7. Иерархия мастеров форм в системе ИСТИНА (фрагмент)

[34] https://github.com/applegrew/django-select2


Авторы. Выберите сотрудников из найденных в системе или добавьте новых

Автор	Похожие сотрудники в системе
Соколов М.Э.	**Соколов М.Э.** // добавить нового сотрудника // затрудняюсь выбрать
Бармин В.В.	**Бармин В.В.** // Бармин А.В. // добавить нового сотрудника // затрудняюсь выбрать
Буданов В.М.	**Буданов В.М.** // добавить нового сотрудника // затрудняюсь выбрать
Галатенко А.В.	**Галатенко А.В.** // Галатенко В. // добавить нового сотрудника // затрудняюсь выбрать
Галатенко В.В.	**Галатенко В.В.** // Галатенко В. // добавить нового сотрудника // затрудняюсь выбрать
Коршунов А.А.	коршунов а.а. // Коршунов А. // **Коршунов Андрей Александрович** // добавить нового сотрудника // затрудняюсь выбрать
Козорезов Ю.Ю.	**Козорезов Ю.Ю.** // добавить нового сотрудника // затрудняюсь выбрать
Подольский В.Е.	**Подольский В.Е.** // добавить нового сотрудника // затрудняюсь выбрать
Садовничий В.А.	**Садовничий В.А.** // Садовничий В.А. // добавить нового сотрудника // затрудняюсь выбрать

Рис. 8. Шаг разрешения неоднозначностей фамилий в мастере форм

трудниками в базе данных. Все проводимые операции записываются в журнал событий. Отдельно следует отметить мастер форм MergeWizardView, который реализует общие функциональные возможности и интерфейс слияния различных объектов. Более подробно он представлен в следующем подразделе.

3.2. Другие функции ядра

В настоящем подразделе рассмотрены механизмы, которые реализуют такие функции системы, как поиск и объединение дубликатов, кэширование данных. Их основной целью является повышение качества и доступности информации в системе.

3.2.1. Слияние различных объектов

Для повышения качества данных в системе присутствует механизм полуавтоматического поиска и объединения похожих объектов, реализованный мастером форм MergeWizardView. В настоящее время он используется для объединения дубликатов конференций. Он может также применяться для устранения дубликатов других объектов в системе, в том числе различных видов деятельности сотрудника, например, статей или книг.

Для удобной работы с этим модулем существует веб-интерфейс, доступный для ответственных лиц от отдельных подразделений организаций, в функции которых входит очистка данных в системе. На первом шаге проводится поиск в базе данных по определенным полям, например, по названию (рис. 9). После этого пользователь выбирает возможные дубликаты объектов и переходит на страницу подтверждения, где может получить более подробную информацию о выбранных объектах (рис. 10). Вид информации зависит от типа объединяемых объектов. В случае конференций решение об объединении объектов происходит на основе информации о годе, месте проведения, числе докладов этой конференции, добавленных в систему, а также известных членах комитета. Данный интерфейс может работать с любыми моделями, которые наследуются от класса MyModel. Для этого нужно задать для него методы, определяющие поля поиска и информацию, отображаемую ответственному лицу. Отображение информации об объектах из базы на странице мастера форм реализовано с использованием сторонней библиотеки DataTables[35].

MergeWizardView реализован таким образом, что объединение любых объектов происходит единообразно и практически не требует изменений с расширением системы и добавлением в нее новых типов объектов. Он значительно упрощает процесс поиска дубликатов и в перспективе значительно повысит качество данных в системе.

3.2.2. Кэширование

Для снижения нагрузки на базу данных используется кэширование различных объектов или части информации о них. Такой подход значительно ускоряет время загрузки некоторых страниц. Кэширование особенно эффективно для относительно редко меняющихся страниц, например, домашних страниц сотрудников. В качестве используемых средств была выбрана связка Memcached[36] + pylibmc[37], которая на момент внедрения кэширования в систему ИСТИНА показывала себя эффективной для приложений, написаных на Python. Механизмы Memcached позволяют хранить кэшируемые значения в выделенной под это оперативной памяти сервера, при этом необязательно того же сервера, на котором расположено приложение на Python. Программное средство Pylibmc обеспечивает доступ к API Memcached в Django. В Django реализована система сигналов, позволяющая получать уведомления о различных событиях, в рассматриваемом случае — об изменении кэшируемого объекта[38].

Инвалидация кэша происходит по истечении заданного срока (в настоящем, через месяц после записи в кэш, это максимум, который позволяет Memcached) или же в случае, если был послан сигнал,

[35] https://datatables.net/
[36] http://memcached.org/
[37] http://sendapatch.se/projects/pylibmc/index.html
[38] https://docs.djangoproject.com/en/dev/topics/signals

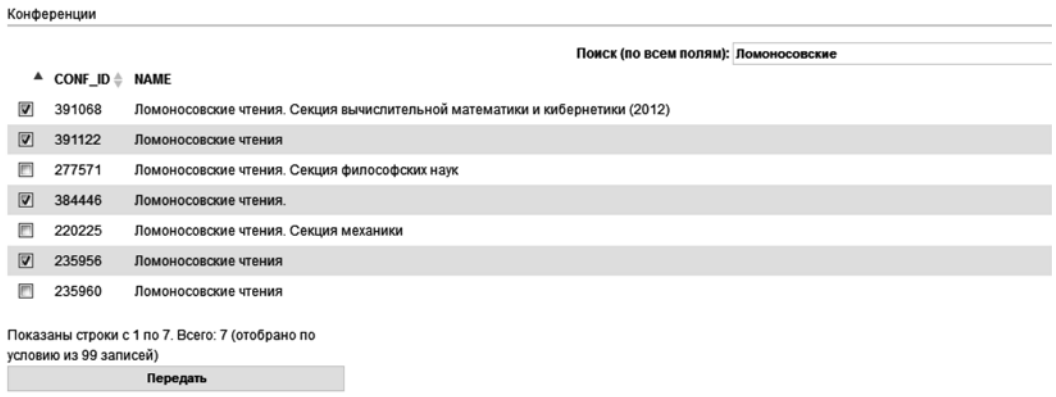


Рис. 9. Поиск похожих объектов

Рис. 10. Страница подтверждения объединения дубликатов (звездочкой обозначена строка, выделенная желтым фоном в цветном варианте)

что кэш устарел и его необходимо инвалидировать. Время, по истечении которого нужно инвалидировать кэш, хранится в переменной CACHE_TIME. Доступ к элементам кэша осуществляется по ключу. При кэшировании частей шаблонов ключ генерируется автоматически при помощи дополнительных аргументов, которые передаются специальному шаблонному тегу {% cache %}.

На настоящее время кэшируются список соавторов и результирующая информация на странице сотрудника или странице пользователя. Для каждого сотрудника кэшируется список соавторов и число его работ. Кэшируются также части шаблона activities.html, отвечающие за отображение деятельности сотрудника. Вся деятельность сотрудника разбита по категориям. Список результатов деятельности в каждой категории каждого сотрудника является отдельным элементом кэша.

Примерами случаев, когда необходимо инвалидировать элемент кэша, являются следующие:

- добавлена, удалена или отредактирована работа, необходимо инвалидировать кэш соответствующей категории у ее авторов;
- добавлено, удалено или отредактировано авторство работы WorkershipModel;
- изменяется имя автора работы;
- изменяется информация о связанном объекте, например, название конференции, на которой был прочтен доклад, в этом случае необходимо обновить кэш для данного доклада.

Заключение

Представлены методы, механизмы и инструментальные средства системы ИСТИНА, с помощью которых реализуется подход, отличающий эту систему от других подобных разработок. В ядре системы заложены возможности расширения спектра видов результатов научной деятельности, которые поддерживаются ею. Базовые модели, формы и мастера форм позволяют сравнительно легко добавлять новые виды результатов, повторно используя общий код. Именно благодаря этим функциям в системе на настоящее время поддерживается 20 видов результатов деятельности. Механизмы и интерфейс слияния объектов в системе, также основанные на едином базовом наборе функций, позволяют пользователям и ответственным

сотрудникам от подразделений самим выполнять очистку данных. Таким образом, повышается точность и корректность данных в системе. Методы и алгоритмы сопоставления и привязки авторов при добавлении работ служат для того, чтобы вопросы синонимии фамилий решались пользователями сразу же при добавлении в систему нового объекта (публикации, патента и др.), а статьи и другие результаты деятельности при этом попадали на страницы только к "нужным" сотрудникам. Таким образом, все представленные выше методы и средства направлены на то, чтобы в системе ИСТИНА содержалась наиболее точная и полная информация о результатах деятельности сотрудников научно-образовательных организаций. Пользователи при этом получают возможность управлять данными о себе и формировать различные отчеты, а руководителям подразделений организаций предоставляются механизмы для анализа деятельности подведомственных им структур и отдельных сотрудников как по статистическим показателям, так и по тематическим направлениям их исследований.

Список литературы

1. **Васенин В. А., Афонин С. А., Козицын А. С.** и др. Интеллектуальная система тематического исследования научно-технической информации (ИСТИНА) // Обозрение прикладной и промышленной математики. 2012. Т. 19, № 2. С. 239—240.

2. **Васенин В. А., Афонин С. А., Голомазов Д. Д., Козицын А. С.** Интеллектуальная Система Тематического Исследования Научно-технической информации (ИСТИНА) // Информационное общество. 2013. № 1—2. С. 21—36.

3. **Афонин С. А., Бахтин А. В.** Построение иерархии понятий на основе лексических шаблонов // Информационные технологии. 2012. № 3. С. 2—7.

4. **Голомазов Д. Д.** Выделение терминов из коллекции текстов с заданным тематическим делением // Информационные технологии. 2010. № 2. С. 8—13.

5. Афонин С. А., Гаспарянц А. Э. Разрешение неоднозначности авторства публикаций при автоматической обработке библиографических данных // Программная инженерия. 2014. № 1. С. 25—29.

V. A. Vasenin, Professor, Head of Laboratory, e-mail: vasenin@msu.ru,
D. D. Golomazov, Senior Reseacher, e-mail: denis.golomazov@gmail.com,
G. M. Gankin, Programmer, e-mail: gregorian21@yandex.ru, Institute of Mechanics Lomonosov Moscow State University

Architecture, Methods and Tools of the Intellectual System of Thematic Analysis of Scientific Information "ISTINA – Science in MSU" Core

The present paper is devoted to a description of architecture and core modules of Intellectual System of Thematic Analysis of Scientific Information (ISTINA, "Truth" in Russian). The system is aimed at gathering, managing, storing, searching, retrieving and analyzing of information that characterizes results of scientific and educational organizations' activity. The system is actively used in Lomonosov Moscow State University. In this paper, the architecture and structure of the core modules are presented. The paper focuses on description of methods and tools of its basic components (the core), including mechanisms of adding new work types and data editing. The paper also describes some practical aspects of the system architecture from a programming point of view.

Keywords: *scientific information, information system, scientific research, performance analysis, bibliographic system, metadata*

References

1. **Vasenin V. A., Afonin S. A., Kozicyn A. S**. et al. Intellektual'naja sistema tematicheskogo issledovanija nauchno-tehnicheskoj informacii (ISTINA). *Obozrenie prikladnoj i promyshlennoj matematiki*. 2012. Vol. 19, N. 2. P. 239—240.

2. **Vasenin V. A., Afonin S. A., Golomazov D. D., Kozicyn A. S.** Intellektual'naja Sistema Tematicheskogo Issledovanija Nauchno-tehnicheskoj informacii (ISTINA). *Informacionnoe obshhestvo*. 2013. N. 1—2. P. 21—36.

3. **Afonin S. A., Bahtin A. V.** Postroenie ierarhii ponjatij na osnove leksicheskih shablonov. *Informacionnye tehnologii*. 2012. N. 3. P. 2—7.

4. **Golomazov D. D.** Vydelenie terminov iz kollekcii tekstov s zadannym tematicheskim deleniem. *Informacionnye tehnologii*. 2010. N. 2. P. 8—13.

5. **Afonin S. A., Gasparjanc A. Je.** Razreshenie neodnoznachnosti avtorstva publikacij pri avtomaticheskoj obrabotke bibliograficheskih dannyh. *Programmnaya ingeneria*. 2014. N. 1. P. 25—29.

УДК 004.4'2

П. А. Ченцов, канд. физ.-мат. наук, науч. сотр., e-mail: chentsov.p@uran.ru,
Институт математики и механики им. Н.Н.Красовского УрО РАН, г. Екатеринбург

Платформа разработки интернет-приложений MVE[1]

Рассмотрена платформа разработки web-приложений MVE (Model/View/Events), построенная на основе свободно распространяемого программного обеспечения PHP, Apache, MySQL. Платформа ориентирована на приложения, взаимодействующие с базами данных. Тем не менее возможно использование платформы MVE с другими источниками данных, такими как файловая система, электронные таблицы. Ключевые особенности платформы: компактность программ и высокий уровень повторного использования кода

Ключевые слова: *интернет-приложение, платформа разработки, автоматическая генерация страниц*

Введение

На этапе становления сети Интернет сайты представляли собой наборы HTML-страниц, связанных гиперссылками. Такие сайты, как правило, содержали статический контент. По сути, это были гипертекстовые документы, находящиеся в свободном доступе. С развитием сети Интернет и информационных технологий появилась потребность в доступе через интернет-браузеры к различным данным. Классический пример — сайт интернет-магазина. Страницы здесь формируются динамически, на основе информации об ассортименте и товарах на складе. Как привило, непосредственно информация хранится в базах данных. Подобные сайты представляют собой серверные программы, формирующие по запросам пользователей HTML-страницы. Клиентские части могут содержать скрипты, написанные, например, с использованием таких технологий, как JavaScript, AJAX, Flash, SilverLight, которые выполняются браузерами. Такие системы называют интернет-приложениями. Интернет-приложения — это не только сайты-интерфейсы баз данных. Это могут быть различные конвертеры, системы обработки файлов, онлайн-текстовые редакторы и т.д. В настоящей статье рассматривается именно аспект работы с данными.

Потребность в интернет-приложениях, предоставляющих доступ к самым различным базам данных, в настоящее время крайне высока. Это магазины, социальные сети, корпоративные сайты, сайты государственных услуг, хранилища файлов и многое другое. Данное обстоятельство свидетельствует об актуальности рассматриваемой темы.

Для реализации подобных сайтов создано большое число систем разработки. К ним можно отнести ASP.NET MVC [1], ASP.NET Forms [2], Zend Framework, Drupal [3]. Как правило, любая система разработки программного обеспечения может быть использована для построения программ, стыкующихся с web-сервером через CGI (*Common Gateway Interface*). И тем не менее многие программисты создают свои платформы, опирающиеся на какой-либо язык программирования. Основная причина здесь в том, что не существует универсальных систем разработки. Что хорошо для одних задач, то может оказаться плохо для других. Одни платформы очень гибки, но требуют от разработчика значительных объемов рутинной работы, что в условиях недостатка времени и должного числа разработчиков может оказаться неприемлемым. Другие платформы, наоборот, высокоуровневые и узконаправленные, и с некоторыми задачами их функциональные механизмы справиться не в состоянии.

В данной работе рассматривается платформа MVE [4] (*Model/View/Events*), созданная автором статьи для разработки интернет-приложений. При создании платформы MVE предпринята попытка автоматизировать многие типовые рутинные функции, возникающие в процессе разработки, а также обеспечить как можно более высокий уровень повторного использования кода.

Постановка задачи

Рассматривается задача отображения и редактирования информации, хранящейся в базе данных, через интернет-браузер. По проектному замыслу система должна состоять из описанных ниже компонентов. Серверная часть: сервер баз данных с развернутой базой и HTTP-сервер. Данное программное обеспечение расположено на одном или разных серверных компьютерах. В качестве клиентской части рассматривается интернет-браузер. Пользователь может запускать браузер на любом компьютере, мобильном устройстве, Smart-телевизоре, при условии наличия доступа к сети Интернет. Пользователь запускает браузер, вводит URL и попадает на страницу, посредством которой он может получать информацию из базы данных, а также создавать новые записи, редактировать и удалять существующие.

Практическая реализация должна быть направлена на формирование HTML-разметки, по возможности без применения JavaScript (за исключением диалогов

[1] Работа выполнена в рамках проектов РЦП-14-ЕИП и РЦП-14-И13.

подтверждения), AJAX, сложных CSS-стилей, всплывающих окон и других технологий, подобных перечисленным. Такие страницы правильно отображаются в любых браузерах и на мониторах с низким разрешением. Основное назначение системы — работа с данными. Система должна правильно выводить большие таблицы с множеством колонок. Как показывает практика, HTML-документы с небольшим числом так называемых современных технологий достаточно хорошо вписываются в окна любых браузеров, причем самых разных размеров. Еще раз подчеркнем тот факт, что система должна работать не в ущерб информативности. Например, если на странице требуется осуществлять выбор из строк большой длины, то следует отдать предпочтение таблице с кнопками и радиобаттонами, нежели выпадающему списку. Пользователь должен получать максимум информации по возможности без сокращений и исключения каких-либо данных.

В базе хранится структурированная информация в виде таблиц. Можно выделить две основных конструкции, посредством которых будет проводиться работа с базой — таблица и экранная форма для работы с записями.

Таблица должна позволять отображать информацию в своих столбцах. Кроме того, должна присутствовать возможность создавать столбцы с кнопками, посредством которых в обработчиках будут выполняться те или иные действия со строками таблиц в базе. Должны быть предусмотрены столбцы, посредством которых будет проводиться редактирование данных непосредственно в ячейках таблицы, а также столбцы, используемые для выделения строк таблицы в целях их групповой обработки. Для создания новых записей и некоторых других целей должна быть предусмотрена возможность создавать отдельно стоящие кнопки до и после таблицы. Для таблиц, содержащих большое число строк, должны быть предусмотрены конструкции постраничного отображения информации в виде кнопок с номерами страниц, располагаемые до и после таблицы.

Форма должна работать с различными полями ввода, к числу которых относятся текстовые поля; выпадающие списки; чекбоксы; радиобаттоны; кнопки; поля разметки. Предполагается, что объект-форма работает с одной записью таблицы (или записями в нескольких таблицах, связанных отношением один к одному).

Как правило, информационные конструкции представляются более сложными, нежели одиночная таблица с данными. Как следствие должна присутствовать возможность вкладывать одни объекты в другие. В качестве примера можно рассмотреть форму с информацией о публикациях. В ней должна быть реализована возможность в любом месте разместить таблицу авторов. Описанные функции позволяют решать значительное число задач по взаимодействию с базами данных.

На реализацию накладываются описанные далее ограничения. Для удобства работы с описанными конструкциями должен быть применен объектно-ориентированный подход. Таблицы и формы — это объекты, внутри которых могут создаваться объекты-колонки и поля. Внутри объектов-полей могут создаваться объекты-валидаторы. Каждый объект системы должен иметь свойства Visible и ReadOnly (видимость объекта и режим "только для чтения"). Генерация непосредственно HTML-страниц должна проводиться автоматически на основе иерархии указанных выше объектов с учетом их видимости и доступности для редактирования. Должен быть предусмотрен комплекс свойств объектов, позволяющих управлять формированием HTML- страниц, отвечающих за стили и разметку.

Объекты-формы и таблицы (за исключением вложенных в другие формы и таблицы) должны рассматриваться как модальные окна. Для пользователя это должно выглядеть следующим образом. Пользователь вводит в браузере URL, после чего открывается страница. Далее он вводит в поля ввода какие-либо значения и нажимает кнопку на странице. После запроса к серверу в браузере отображается другая страница, но с тем же URL. При нажатии на этой новой странице кнопки, предназначенной для возврата к первой, в браузере опять появляется первая страница, причем все введенные в нее пользователем до этого данные должны быть восстановлены в своих полях.

Основные принципы работы через браузер

В классической программе, как правило, интерфейс функционирует в том же процессе, что и алгоритмы работы с данными, что существенно упрощает работу. При этом описывается интерфейс. В обработчиках, связанных с элементами управления, выполняется программный код, тем или иным способом модифицирующий данные.

В случае web-приложений все несколько сложнее. Здесь механизмы обработки данных отделены от интерфейса. Когда пользователь вводит URL в браузере, происходит запрос на HTTP-сервер. Сервер возвращает страницу, расположенную по указанному адресу. Страница содержит элементы ввода и кнопки. Пользователь вводит значения и нажимает кнопку на странице. Для кнопки указан URL-адрес страницы, серверные скрипты которой должны выполнить обработку данных. На сервер отправляется запрос, вместе с которым введенные пользователем данные и имя кнопки уходят в специальных post-параметрах. Массив этих post-параметров доступен в серверном языке программирования. Сервер, получив эти данные, выполняет скрипты указанной для кнопки страницы, которые обрабатывают введенные данные. После этого страница возвращается пользователю. Отсюда следует, что послав пользователю страницу, сервер о ней "забывает". Все необходимые для обработки данные должны прийти назад в ходе обратного запроса в post-параметрах. Следует заметить, что конечно есть данные, получаемые через куки и сессии. Однако они не являются данными отдельных страниц и поэтому здесь не рассматриваются.

В представленном выше случае все выглядит достаточно просто. Однако, что будет происходить, если новая страница также содержит элементы ввода, обработка которых зависит от данных первой страницы?

В такой ситуации при формировании второй страницы сервер должен внести в ее код специальные скрытые поля HTML-формы (тег <INPUT TYPE=”hidden”...>). Далее эти конструкции будем называть просто скрытые поля страницы, так как любая страница в MVE содержит только одну HTML-форму (под формой в данном случае подразумеваем HTML-конструкцию, тег <FORM>). В эти скрытые поля помещаются требуемые данные первой страницы. Второй раз сервер получит запрос, содержащий не только имя нажатой кнопки и введенные пользователем в поля новой страницы значения, но и данные первой страницы (также в post-параметрах).

В системе MVE применяются следующие особенности. Для каждой кнопки в качестве обработчика указывается та же самая страница. Таким образом, в ходе запросов изменяется только отображаемое в браузере содержимое, но URL остается прежним. Для реализации режима модальных объектов используются скрытые поля. Данные неактивных в текущий момент форм и таблиц, не видимых в окне браузера, всякий раз на сервере сохраняются в скрытых полях формируемой для пользователя страницы (приходя на сервер в post-параметрах). Если для объекта поле ввода (речь идет об объектной модели MVE) свойство ReadOnly имеет значение true, то элемент ввода (тег <INPUT>), способный отправить на сервер значение, на странице не создается, вместо него выводится статический текст. Для того чтобы сервер получил значение такого поля при обратном запросе, для него также создается скрытое поле. Отмеченное справедливо и для свойства Visible.

Основные конструкции MVE

В соответствии с постановкой задачи платформа MVE построена на основе объектно-ориентированного подхода. Как отмечалось ранее, основные конструкции для работы с данными это формы и таблицы — два основных класса MVE. Данные классы далее будем называть классами юнитов, а объекты этих классов — юнитами. Программист должен создавать свои классы юнитов, наследуя их от одного из указанных классов, чтобы в конструкторе сформировать структуру вложенных объектов и переопределить обработчики событий в соответствии с решаемыми задачами. Каждый юнит состоит из следующих частей: модель; параметры модели; объектная структура юнита; обработчики событий. Кроме того, табличный юнит содержит так называемую редактируемую модель.

Модель — набор данных, с которыми работает юнит. Для юнита-таблицы модель представляет собой таблицу — двумерный массив PHP. Строки нумеруются числовыми индексами и представляют собой ассоциативные массивы, в которых индексы — имена колонок таблицы, а значения — значения ячеек таблицы.

Для юнита-формы модель представляет собой ассоциативный одномерный массив, в котором индексы — имена полей модели, а значения — значения данных полей.

Модель не обязательно должна содержать только ту информацию, которая была извлечена из базы данных. Некоторые колонки и поля могут быть вычисляемыми. Модель юнита-формы строится либо в специальном обработчике OnCreateModel, либо активирующим юнитом (механизм активации и стек юнитов будут представлены далее) и автоматически восстанавливается в ходе запросов к серверу. Модель юнита-таблицы строится всякий раз в обработчике OnCreateModel.

Кроме простой табличной модели юнит-таблица содержит еще и редактируемую модель. Это также двумерный массив со структурой, похожей на структуру модели. В нем хранятся колонки с измененными значениями отдельных редактируемых колонок модели. В редактируемую модель входят также идентифицирующие колонки. Эти колонки позволяют однозначно идентифицировать строки и установить взаимно-однозначное соответствие между строками этих двух моделей. Такой набор колонок (предназначенных для идентификации строк) будем называть идентифицирующими колонками. Данный набор, как правило, соответствует первичному ключу одной из таблиц. Однако MVE может работать и с другими источниками данных, поэтому термин "первичный ключ" не используется. Редактируемая модель восстанавливается при запросах к серверу. Таким образом, в обработчиках можно написать программный код, который будет сохранять в базу данных введенные в редактируемую модель значения (в этом помогут значения из идентифицирующих колонок).

Параметры модели для обоих типов юнитов представляют собой одномерный ассоциативный массив. Индексы в нем — имена полей. Особенность данного массива заключается в том, что он автоматически восстанавливается при запросах к серверу (например, при нажатии на кнопку формы). В нем можно размещать любые вспомогательные данные, которые могут использоваться в обработчиках событий. Пример — в случае с таблицей авторов публикации в параметрах ее модели можно хранить идентификатор публикации.

В соответствии с постановкой задачи, непосредственно HTML-страница должна генерироваться автоматически на основе объектной модели. Внутри юнита-формы можно создавать специальные объекты: поля ввода; контейнеры вложенных юнитов; чекбоксы; выпадающие списки; поля для выбора радиобаттонами; поля разметки; кнопки. В дальнейшем такие объекты будем называть элементами юнита. Для каждого элемента, представляющего информационное поле (текстовые поля, выпадающие списки и т. д.), необходимо явно указать имя связанного с данным элементом поля модели.

Внутри юнита-таблицы можно создавать элементы — колонки таблицы, кнопки, контейнеры вложенных юнитов. Для элементов информационных колонок также нужно указать имена связанных с ними колонок модели. Исключение составляют информационные колонки, предназначенные для редактирования таблицы — они работают и с обычной, и с редактируемой моделями таблицы. Элементу такой колонки кроме имени колонки модели требуется еще и имя колонки редактируемой модели, в которую будут помещаться измененные значения. В элементах "поля формы" и эле-

ментах "редактируемые столбцы таблицы" можно создавать объекты-валидаторы.

Каждый объект MVE, начиная с юнита, имеет метод прорисовки. Внутри метода прорисовки юнита активируется прорисовка всех вложенных элементов, которые, в свою очередь, активируют прорисовку валидаторов.

В MVE каждый объект, включая любой юнит, идентифицируется посредством текстового имени. Можно получить указатель на юнит, зная его имя. Можно найти любой элемент юнита по имени. По имени можно также найти любой валидатор элемента. Эти имена участвуют в формировании имен тегов <INPUT...> и скрытых полей (<INPUT TYPE=”hidden”...>).

Так как HTML-страница формируется автоматически, то, очевидно, следует иметь способы непосредственного воздействия на получаемое содержимое. Для этого используют так называемые стилевые свойства — блоки текста, которые помещают в заданные участки генерируемого HTML-кода. Для наглядности рассмотрим упрощенный пример. Пусть есть объект, реализующий работу с полем ввода. Предположим, что у него есть три (на самом деле их больше) стилевых свойства: текст первого свойства помещается перед тегом <INPUT...>, текст второго свойства — внутри тега <INPUT...>, текст третьего свойства — после тега <INPUT...>. Это может быть любой текст, включая указание класса CSS (во втором свойстве). Тексты перед и после поля ввода могут, например, обрамлять данное поле тегами параграфа или ячейки таблицы. Для всех этих свойств в MVE существует предустановленный шаблон. Можно создавать новые шаблоны или задавать для отдельных объектов стилевые свойства напрямую. Еще раз подчеркнем, что эти шаблоны, с одной стороны, могут содержать только имена CSS классов, а с другой стороны, могут использоваться для построения достаточно сложной автоматической разметки.

В системе предусмотрено несколько типов основных событий. К ним относятся описанные далее.

- OnPreCreateModel — обработчик события, генерируемого после восстановления данных юнитов, но перед валидацией данных и вызовами событий от кнопок.
- OnRowButtonClick — обработчик события нажатия на кнопку в строке таблицы. Получает параметром имя кнопки и значения идентифицирующих колонок строки, в которой было выполнено нажатие кнопки.
- OnButtonClick — обработчик события нажатия на кнопку юнита-формы или отдельно стоящую кнопку юнита-таблицы. Получает параметром имя нажатой кнопки.
- OnCreateModel — обработчик события построения модели. Как правило, используется для построения модели табличных юнитов.
- Обработчик события прорисовки юнита OnDraw. В обычных условиях не требует переопределения. Следует переопределять только в случае крайней необходимости, если требуется получить особенную страницу сложной конструкции.

Отметим, что обработчики указаны в последовательности, в которой генерируются события.

Принципы функционирования юнита

Для каждой страницы, взаимодействующей с данными путем использования механизмов MVE, задан начальный юнит. Он обязательно должен создаваться в PHP-скрипте в начале страницы. При этом ему сразу должно быть присвоено уникальное имя. Когда пользователь вводит в браузере URL, происходит запрос к серверу, и начальный юнит генерирует HTML-код страницы.

Перед формированием возвращаемого пользователю текста страницы вызывается обработчик OnPreCreateModel, хотя на начальном этапе он не имеет какого-либо существенного значения (более подробные сведения о нем будут представлены далее). После этого вызывается обработчик OnCreateModel. Для таблиц в данном обработчике всякий раз строится модель. Для форм внутри OnCreateModel можно, например, вычислить значения расчетных полей. Возможен также вариант поведения, когда в OnCreateModel строится модель юнита-формы. Модель в этом случае должна строиться один раз, далее она будет автоматически восстанавливаться при запросах к серверу.

После перечисленных выше действий вызывается обработчик OnDraw, внутри которого автоматически осуществляется построение страницы с использованием объектной структуры юнита (его элементов и валидаторов). Элементы юнита (текстовые поля ввода, чекбоксы, группы радиобаттонов, кнопки и т.д.), имеющие значение true свойства Visible и false свойства ReadOnly, формируют на странице теги <INPUT...>. Имена этих тегов будут формироваться на основе специального префикса, имени юнита и имени связанного с элементом поля модели. Строка формируется в специальном формате так, чтобы указанные части можно было из нее извлечь. Отметим данное обстоятельство, так как в дальнейшем эти имена позволят системе восстанавливать значения полей модели и узнавать о нажатии кнопок.

В конструкторе юнита или в обработчике OnCreateModel в параметры модели могут быть записаны какие-либо данные. По завершении формирования визуальной части юнита на странице выполняется процедура сохранения данных, не связанных с полями ввода. Эта процедура очень важная, так как с ее помощью в страницу будут внесены все необходимые впоследствии для сервера данные. Для каждого параметра модели создается скрытое поле (тег <INPUT TYPE=”hidden”...>), чтобы сервер мог получить эти значения при обратном запросе в ходе взаимодействия пользователя с элементами страницы. Имя каждого такого скрытого поля включает префикс, имя юнита и имя параметра модели. Кроме того, в скрытые поля попадают те поля модели юнита-формы, которые не привязаны к элементам юнита, или их элементы неактивны (свойство ReadOnly имеет значение true), или они скрыты (свойство Visible имеет значение false). Имена данных скрытых полей формируются также из префикса, имени юнита и имени поля модели. Следует отметить, что скрытые поля создаются и для свойств Visible и ReadOnly-юнита,

а также для всех его элементов и валидаторов. Для юнита таблицы, содержащего элементы колонки, предназначенные для редактирования, в последующих обращениях к серверу для каждой строки редактируемой модели тоже будут создаваться скрытые поля. Изначально, при первом обращении к странице, редактируемая модель еще пуста.

После перечисленных выше действий страница возвращается пользователю. Пользователь вводит в поля ввода необходимые данные и нажимает кнопку на странице. Браузер формирует обратный запрос к серверу. Данный запрос содержит post-параметры. В эти post-параметры браузером помещаются: название нажатой кнопки; все поля ввода (пары имя—значение, введенное пользователем в поле ввода на странице); все скрытые поля (пары имя—значение). Как было отмечено ранее, в MVE за обработку отвечает та же самая страница. Получив запрос, север начинает выполнение скриптов этой страницы. Следовательно, в ней создается тот же самый объект-юнит. Однако изначально он не содержит никаких данных. Эти данные требуется восстановить. Для этого, обнаружив post-параметры, сервер запускает процедуру восстановления данных. Система автоматически по именам находит в массиве post-параметров все данные юнита и помещает их в соответствующие информационные конструкции. По завершении данной процедуры сервер получает объект-юнит с теми данными, которые были заложены в него при первом создании страницы, а также данные, которые пользователь ввел в поля ввода. К этим данным можно обращаться напрямую, через массивы модели и параметров модели.

Дальнейшие действия выполняет обработчик OnPreCreateModel. К этому моменту еще не осуществлялась валидация введенных данных, обработчики кнопок еще не выполнялись и не внесли никаких изменений. С учетом этих обстоятельств можно написать код, выполняющий требуемые действия над только что полученными, но еще не обработанными данными.

Далее выполняется автоматическая валидация полей. Для каждого элемента-кнопки можно указать набор элементов, для связанных полей модели которых требуется валидация (или другой набор элементов, для которых наоборот не следует выполнять валидацию). Для остальных элементов, даже содержащих валидаторы, проверка значений проводиться не будет. Более подробная информация о процессе валидации будет представлена далее.

После процесса валидации полей происходит вызов обработчика кнопок. Для таблицы это обработчики OnRowButtonClick или OnButtonClick, для формы — только OnButtonClick. Внутри этих обработчиков уже доступна информация о том, успешна валидация данных или нет. Здесь, как правило, проводится создание новых записей, редактирование и удаление существующих. Важно отметить, что для табличных юнитов на данном этапе модель еще не создана. Все необходимые данные приходят в обработчики параметрами (для OnRowButtonClick это имя кнопки и значение идентифицирующих колонок выбранной кнопкой строки, для OnButtonClick — имя кнопки). Тем не менее при наличии редактируемых столбцов редактируемая модель доступна — она восстанавливается в ходе процедуры восстановления данных юнита.

Далее вызывается обработчик OnCreateModel, назначение которого отмечено ранее. После создания модели вызывается обработчик OnDraw, выполняющий прорисовку юнита. В конце опять выполняется указанная ранее процедура сохранения данных в скрытые поля.

Таким образом, программист полностью избавлен от рутинной работы с post-параметрами, которая в больших проектах может оказаться значительной. Более того, совершенно не заметен отрыв интерфейса от сервера. Используются обычные принципы работы с объектной структурой. Например, пусть есть юнит-форма с элементом — строкой ввода. Чтобы получить или записать значение строки в коде обработчика, нужно напрямую обратиться к ячейке модели юнита с заданным именем. Если по каким-то причинам требуется скрыть поле на странице, достаточно присвоить значение false свойству Visible элемента данного поля ввода. Несмотря на то что поле ввода на странице пропадет, в модели оно останется. Значение в этом случае будет восстанавливаться при всяком обращении к серверу. Объектная модель позволяет решать много самых разных задач, о чем будет упомянуто далее.

Стек юнитов MVE

Технология разработки с использованием MVE в самом общем ее представлении похожа на работу с модальными окнами, например, на разработку Windows-приложений. Конкретная страница с ее URL выступает как некое рабочее пространство, в котором происходит работа с набором юнитов. Как было отмечено ранее, на странице есть начальный юнит, формирующий содержимое страницы при переходе по ссылке страницы. Пользователь при этом взаимодействует с элементами страницы. В результате формируется обратный вызов к серверу. На сервере выполняются обработчики событий. В обработчиках событий можно сделать модальным другой юнит. После ответа сервера в браузере будет отображаться страница, сформированная этим вторым юнитом, но URL страницы останется прежним. Локальные данные первого юнита будут сохранены – к нему можно будет вернуться, "закрыв" второй юнит. Из второго юнита можно сделать модальным третий и т.д.

Главное отличие подхода MVE от классического стиля работы с окнами состоит в том, что при его реализации юнит может выполнять как функцию отдельного модального диалога, так и оконного элемента, контрола, который располагается среди содержимого другого юнита. Например, с позиций C# это могло бы выглядеть так: есть класс, способный порождать объекты, эти объекты без изменений могут использоваться и как полноценная оконная форма, и как контрол (элемент формы), который можно расположить на любой другой форме. Данный пример приводится для наглядности, на самом деле в C# этого делать нельзя. Отмеченные возможности очень удобны с точки зрения повторного использования кода, а так-

же централизованного управления содержимым страницы и структурирования программы. Далее рассмотрим как именно это реализуется.

Все юниты, которые могут быть задействованы программистом в пределах данной страницы, должны быть созданы в начале этой страницы. В начале страницы подключаются файлы с классами юнитов и создаются объекты юнитов. Каждому юниту дается уникальное имя. Указатель на каждый юнит при создании автоматически помещается системой в специальный массив, обеспечивая возможность поиска по имени. В дальнейшем программист будет работать именно с этими объектами, а не с HTML-страницами или post-параметрами. Чтобы система начала работать с юнитом, следует выполнить его активацию. Активация юнита сходна с вызовом метода DoModal объекта класса CDialog библиотеки MFC в системе разработки Microsoft Visual C++ или ShowDialog объекта класса Form в языке C#.

Вся работа начинается с активации начального юнита при первом обращении к странице. Данный вопрос не рассматривался в предыдущем разделе, чтобы не перегружать читателя информацией. В ходе активации имя юнита добавляется в хвост специального массива имен юнитов стека. Данный массив при каждом запросе передается серверу в виде post-параметров. Для этого в конце прорисовки страницы создаются скрытые поля. Первое поле содержит число записей массива и имеет фиксированное зарезервированное имя. Другие поля содержат значения ячеек массива с именами юнитов стека. Имена этих полей складываются из фиксированного префикса и индекса в массиве. Получив запрос, сервер сначала извлекает из post-параметров размер массива имен юнитов стека, после чего восстанавливает требуемое число ячеек этого массива, обращаясь к post-параметрам, имя которых складывается из префикса и индекса считываемого имени юнита стека.

После того как массив имен юнитов стека восстановлен, для каждого имени из этого массива проводится поиск соответствующего объекта юнита и выполняется процедура восстановления его данных из post-параметров. Осуществляется также поиск всех вложенных в него юнитов и восстановление их данных (механизм вложения юнитов будет рассмотрен далее). После завершения всех этих действий в распоряжении программиста оказываются все юниты стека с восстановленными информационными конструкциями, вплоть до видимости и доступности отдельных элементов и валидаторов.

Далее для последнего юнита стека выполняются описанные в предыдущем разделе действия (за исключением уже проведенного восстановления данных). В ходе этих действий создается возвращаемая пользователю страница. Эти действия выполняются также для вложенных юнитов (в последний юнит стека), которые в стеке не присутствуют (вложенный юнит считается частью родительского). Остальные юниты стека невидимы пользователю, никакие их конструкции не визуализируются в окне браузера. Тем не менее в конце прорисовки страницы для всех таких юнитов массивы моделей, массивы редактируемых моделей, массивы параметров моделей, значения свойств Visible и ReadOnly самих юнитов и их элементов с валидаторами сохраняются в скрытые поля. Имена этих полей имеют специальные форматы и содержат фиксированные префиксы, имена юнитов, элементов, валидаторов информационных конструкций и т. д. Как нетрудно догадаться, делается это для того чтобы при следующем запросе к серверу данные всех указанных юнитов стека были снова восстановлены.

Независимость юнитов и относительные имена

В реальных задачах для эффективной работы с данными, как правило, необходима комбинация юнитов. Например, внутри юнита-формы данных публикации требуется разместить юнит-таблицу ее авторов. Для этого в обоих типах юнитов предусмотрены специальные элементы-контейнеры, которые можно создавать среди других элементов (таких как элементы кнопок, полей ввода и т.д.). В свойствах контейнера указывается имя вкладываемого юнита или указатель на него.

Взаимодействие между юнитами реализуется посредством прямых обращений к свойствам, методам и полям других объектов юнитов. Иными словами, внутри обработчиков все юниты прозрачны и доступны друг для друга. В системе присутствует функция, которая возвращает по имени указатель на юнит. Как следствие, даже не появляется необходимость хранить указатели в переменных.

Для иллюстрации представленного выше подхода вернемся к юниту-публикации и вложенному в него юниту-таблице авторов. Таблице авторов обязательно нужен идентификатор публикации, чтобы в обработчике OnCreateModel правильно построить модель. Для этого в OnCreateModel таблицы авторов можно запросить у системы по имени указатель на юнит-публикацию, а также обратиться напрямую к его модели, а именно к ячейке массива модели, содержащей идентификатор публикации.

Рассматриваемый подход очень удобен и нагляден. Однако все его положительные качества оказываются под вопросом, когда возникает перспектива одновременного использования в пределах стека нескольких экземпляров юнит-публикации. Напомним, у каждого из них уникальное имя и в каждый вложена таблица авторов, имя которой также должно быть уникальным. Этот факт означает, что для каждой публикации следует создать юнит-таблицу авторов, присвоить ей уникальное имя и связать с контейнером внутри публикации. Вместе с тем на практике число вложенных юнитов может быть достаточно большим, как следствие появится рутинная работа, с которой и пытаемся бороться. Другой, менее очевидный вопрос обусловлен связанностью такой конструкции. Если появляется потребность автономно использовать юнит-таблицу авторов, то сделать это не удастся. Причина в том, что контейнер получает идентификатор публикации из родительской юнит-публикации и не может функционировать автономно от нее. Однако хотелось бы иметь та-

кую иерархию юнитов, в которой каждый юнит мог бы функционировать отдельно вместе со своими вложенными юнитами.

Решить все перечисленные задачи призвана система относительных имен. Имя юнита — свойство с методами get/set (в MVE практически всюду используются такие свойства). Set-метод свойства позволяет задать имя юнита. Если режим относительных имен выключен, то get-метод возвращает то же самое значение. Если режим включен, возвращается составное имя, которое складывается из имени родительского юнита, знака подчеркивания, и указанного через set-метод имени текущего юнита. Следует заметить, что имя родительского юнита также может быть относительным и состоять из нескольких частей.

Все перечисленные выше возможности позволяют создать вложенный юнит в конструкторе родительского, дать ему имя в контексте родительского юнита, включить режим относительного имени и поместить в элемент-контейнер. Вернемся к приведенному примеру. Пусть имя юнит-публикации — Publication. Указанное в конструкторе Publication имя юнит-таблицы авторов — Authors. Таким образом, относительное имя таблицы авторов будет Publication_Authors. При создании экземпляра Publication таблица авторов будет создаваться в конструкторе юнита-публикации и получать правильное имя. Кроме того, ссылка на таблицу авторов здесь же в конструкторе будет назначаться контейнеру. Таким образом, программисту достаточно создать заданное число юнитов-публикаций, при этом не заботиться о вспомогательных, используемых ими юнитах.

Помимо вложенных,юнит может работать и с другими юнитами. Возвращаясь к рассматриваемому примеру, авторов в таблицу нужно как-то добавить. Для этого предлагается использовать юнит-таблицу сотрудников. Непосредственно для выбора авторов ее придется активировать. Как следствие, первые два юнита будут скрыты в стеке, их визуальное содержимое уйдет со страницы, сменившись полной таблицей сотрудников с кнопками и чекбоксами выбора. Вместе с тем хотелось бы, чтобы эта самая таблица сотрудников создавалась в конструкторе таблицы авторов и тоже получала относительное имя. Такой прием обеспечит полностью независимые комплекты юнитов. Поэтому в MVE относительное имя можно задавать не только вложенным юнитам, но и используемым отдельно, активируемым из текущего юнита. Такие (используемые отдельно) юниты будем называть подчиненными. Для этого нужно задать подчиненному юниту указатель на тот юнит, который считается родительским, и включить режим относительного имени. Так, в рассматриваемом примере относительное имя таблицы сотрудников может быть Publication_Authors_PersonSelect.

Возникает вопрос правильного взаимодействия юнитов в подобной иерархии. Это взаимодействие можно разделить на две части — взаимодействие с нижестоящими и стоящими выше в архитектурной иерархии юнитами.

При взаимодействии с нижестоящими юнитами родительский юнит должен поместить все необходимые данные в параметры модели вложенного или подчиненного юнита. Иными словами, вложенный или подчиненный юнит должен функционировать из предпосылки, что все необходимые данные у него уже есть, и решать только свою задачу. В нашем примере юнит-публикация обязан в своем обработчике OnCreateModel поместить идентификатор публикации в параметры модели таблицы авторов. Следовательно, при вызове OnCreateModel таблицы авторов идентификатор публикации у нее уже будет. С учетом того что параметры модели сохраняемые, этот идентификатор у таблицы авторов будет и в дальнейшем, вплоть до выхода самой таблицы авторов или ее родительского юнита из стека.

Для взаимодействия с вышестоящими юнитами также можно использовать параметры модели. Правда это не всегда удобно, так как данные родительскому юниту могут потребоваться однократно, а в параметрах модели они будут существовать вплоть до выхода вышестоящего юнита из стека. Кроме того, как правило, родительскому юниту требуется какая-либо реакция на действия вложенного или подчиненного юнита. Таким образом, предлагается использовать специальные методы. Будем называть их методами обратного вызова. Иными словами, если родительский юнит использует вложенный или подчиненный юнит заданного класса, то он обязан иметь специальный метод (или методы) обратного вызова, принимающий заданный набор параметров. В случае необходимости взаимодействия с родительским дочерний юнит просто вызывает данный метод родительского юнита, передав ему нужные параметры (указатель на родительский юнит есть и у вложенного, и у подчиненного юнитов). Таким образом, родительский юнит принимает данные от дочернего юнита параметрами и выполняет требуемые действия по обработке этих данных. Вернемся к нашему примеру. Так как таблица авторов работает с таблицей выбора сотрудников, то она (таблица авторов) обязана иметь метод обратного вызова, который принимает параметром массив идентификаторов выбранных сотрудников. Данный метод будет выполнять добавление этих сотрудников в авторы публикации в базе данных. Таблице сотрудников, со своей стороны, совершенно все равно, для каких целей она проводит выбор сотрудников и какой юнит ее об этом запрашивает. Пользователь отмечает галочками требуемых сотрудников и нажимает кнопку выбора на странице. После этого на сервере юнит-таблица сотрудников просто вызывает указанный выше метод родительского юнита (который обязан быть, иначе нет смысла работать с таблицей сотрудников), передавая ему массив идентификаторов выбранных сотрудников. Внутри этого метода юнит-таблицы авторов сотрудники из массива добавляются в авторы публикации. Далее таблица сотрудников "закрывается", уходит из стека, возвращая пользователя к странице публикации.

С учетом представленных выше соображений возникает вопрос: но ведь таблица авторов должна как-то менять свой вид в зависимости от целей, в которых она используется? Вышестоящий юнит может в своем обработчике OnCreateModel поместить информаци-

онную строку в массив параметров модели дочернего юнита (в нашем примере это может быть строка "Укажите авторов публикации"). Дочерний юнит в своем обработчике OnCreateModel просто копирует это значение в свой заголовок. Можно предложить и другие схемы, но оставим их на рассмотрение читателю.

Теперь можно сформулировать основной принцип независимости юнитов. *Каждый юнит в иерархии может быть автономно использован вместе со своими вложенными и подчиненными юнитами, со всем нижестоящим деревом.*

Объектная модель дает богатые возможности в плане изменения состава страницы, формируемой юнитом. Иными словами, в MVE один и тот же юнит может порождать множество сущностей. Например, юнит-публикация может работать в разных режимах — формировать страницы редактирования и отображения данных публикации, которые могут делиться на подробные и краткие, которые, в свою очередь, в зависимости от прав доступа сотрудника могут скрывать часть информации (например, доступ к файлу публикации).

Подводя итог изложенным выше соображениям, можно сказать, что, с одной стороны, можно использовать один и тот же юнит для любых задач, связанных с описываемой им информационной конструкцией, а с другой стороны, можно включать его в функциональные возможности любого другого юнита, также решающего любые связанные с ним задачи. Все это обеспечивает очень высокий уровень повторного использования кода.

Валидация вводимых данных

Важная составляющая подобных информационных систем — проверка вводимых пользователем данных. В MVE для этих целей предназначен специальный класс валидаторов. Его объекты, валидаторы, могут создаваться внутри элементов. Причем один элемент может содержать несколько валидаторов. Валидаторы могут проверять значение связанного с элементом поля модели на его принадлежность к целым числам и числам с плавающей запятой, а также осуществить проверку попадания этого значения в заданный диапазон. Кроме того, возможна проверка ввода даты и времени, строки, длина которой должна попадать в указанный диапазон. Есть режим проверки с использованием регулярных выражений.

Взаимодействие пользователя с системой происходит посредством кнопок страницы. Далеко не всегда при нажатии на ту или иную кнопку следует выполнять валидацию всех элементов юнита. Каждому элементу кнопки можно назначить массив имен элементов с валидаторами, проверка которых должна проводиться при нажатии на данную кнопку. Иногда на больших страницах проще указать набор элементов, валидацию которых не следует проводить в отличие от всех остальных. Такая возможность тоже существует. Если валидация требуется для всех элементов, то описанные выше массивы имен указывать не следует.

Важно отметить тот факт, что все проверки осуществляются системой автоматически. В случае неправильного ввода возле соответствующего управляющего элемента на странице появляется предустановленное в валидаторе сообщение. В обработчиках кнопок доступна информация о том, прошла валидация или нет.

Особенности применения объектной модели

Около HTML-кода, создаваемого каждым элементом, для которого обязателен ввод значения, следует рисовать значок обязательного ввода — обычно это красная звездочка. Если на странице есть такие элементы ввода, то в начале или в конце страницы обязательно должна идти расшифровка указанного значка. Ранее отмечалось, что один юнит может порождать множество сущностей — страниц, на которых одни элементы доступны, для других включается режим ReadOnly, третьи вообще оказываются невидимыми. Получается, что изменяя значения свойств ReadOnly и Visible, программист должен не забывать включать или выключать валидаторы, а также выводить значок обязательного ввода. Кроме того, следует контролировать, есть ли в текущей версии юнита элементы с обязательным вводом значения, и при необходимости выдавать расшифровку значка обязательного ввода. Все это рутинная работа, которая не просто занимает много времени — в ней еще и легко сделать ошибки. При использовании возможностей MVE вся эта работа возложена на систему. Встроенные механизмы проверяют видимость элемента, и если он невидим, отключаются и валидаторы. Для видимых элементов проверяется свойство ReadOnly. По умолчанию, если свойство ReadOnly элемента имеет значение true, все его валидаторы отключаются, значок обязательного ввода не отображается. При появлении необходимости это поведение можно изменить посредством специального свойства. Если у видимого и доступного для редактирования элемента есть хотя бы один валидатор, не допускающий пустого значения связанного поля модели, то при формировании страницы система автоматически дорисовывает значок обязательного ввода возле тега <INPUT...>. Если в юните обнаружены элементы, для которых система приняла решение об обязательном вводе, в начале страницы автоматически отображается информация с расшифровкой значка обязательного ввода.

Следующая особенность, которая требует рассмотрения, касается режима ReadOnly. Например, присвоим значение true свойству ReadOnly юнит-публикации в нашем примере. Хотелось бы, чтобы все его элементы, а также вложенный юнит-таблица авторов тоже перешли в режим отображения. По умолчанию все элементы системы и вложенные юниты наследуют свойство ReadOnly вышестоящего объекта. Однако такое поведение можно изменить при помощи специального свойства. Данная функция крайне полезна — изменив одну строчку можно, например, получить вместо страницы редактирования публикации страницу отображения данных публикации.

В свете наследования режима ReadOnly возникает потребность в еще одном нововведении — автоматическом сокрытии отдельных элементов в режиме ReadOnly или, наоборот, отображение их только в этом случае. Для указания такого поведения элементы

имеют специальное свойство. Рассмотрим, почему это так важно. В нашем примере таблица авторов публикации имеет кнопку добавления авторов. Кроме того, есть колонка с кнопками удаления авторов. Так вот, для этих элементов можно включить автоматическое сокрытие в режиме отображения. Если для юнита-публикации включится режим ReadOnly, юнит-таблица авторов тоже наследует этот режим, и указанная кнопка и колонка исчезнут из таблицы авторов автоматически. Другой пример — таблица публикаций. В режиме редактирования в ней должны быть колонки с кнопками изменения данных публикации и удаления записи, а под таблицей должна быть кнопка создания новой записи. В режиме отображения все эти элементы должны уйти со страницы, а вместо них должен появиться столбец с кнопками отображения данных публикации. Очень удобно отметить такое поведение для элементов прямо при их конфигурировании. Это намного удобнее и нагляднее, нежели в OnCreateModel проверять свойство ReadOnly юнита, и в зависимости от значения изменять вручную свойство Visible нужных элементов.

При описании взаимодействия с нижестоящими юнитами отмечалась передача значений в их параметры модели. Данная передача предполагается в OnCreateModel, но это — отдельный программный код. Для удобства возможностями MVE предусмотрено описание передачи значений при конфигурировании элемента-контейнера. Делается это посредством метода, принимающего четыре параметра: источник (модель или параметры модели) текущего юнита; имя поля модели или параметра модели; получатель (модель или параметры модели) для вложенного юнита; имя поля модели или параметра модели вложенного юнита. Эта функция значительно повышает наглядность программы.

Другая особенность рассматриваемого подхода состоит в том, что если элемент "текстовое поле" перешел в режим отображения и связанное с ним поле модели содержит пустую строку, то в большинстве своем данное поле не следует отображать на формируемой странице. Кроме того, если для вложенной таблицы включен режим ReadOnly, и она не содержит строк, то ее присутствие на странице тоже представляется ненужным — отобразятся только заголовки столбцов. Все это лишь загромождает страницу ненужными конструкциями, не содержащими информации. В MVE по умолчанию в режиме ReadOnly элементы "текстовое поле с пустой строкой", а также таблицы с пустой моделью не формируют HTML-кода, избавляя программиста от необходимости делать это вручную. При появлении необходимости такое поведение можно изменить посредством специального свойства.

Для элементов кнопок можно включить режим подтверждения, указав в специальном свойстве текст сообщения для диалога подтверждения. В этом случае при нажатии на кнопку пользователю выдается диалог подтверждения. Запрос уходит на сервер только в том случае, если пользователь подтверждает правильность нажатия.

Если пользователь нажал кнопку в строке таблицы, то далее эта строка будет автоматически подсвечена, и страница будет прокручена так, чтобы строка попала на экран. Приведем пример: пользователь нажал кнопку редактирования в таблице публикаций, активировав тем самым юнит-публикацию, по завершении редактирования юнит-публикация "закрывается", происходит возврат к таблице публикаций, в которой строка только что отредактированной публикации будет подсвечена и гарантированно попадет на экран. Это особенно удобно, если нужно последовательно обрабатывать записи.

Еще одно нововведение касается сокрытия отдельных элементов таблицы в случае, если число строк станет меньше заданного значения. Дело в том, что обычно такие элементы, как кнопка добавления записи и кнопки страниц, при постраничном выводе дублируются до и после таблицы. Однако выводить их до таблицы в случае, если она содержит несколько строк, нет смысла — они лишь будут загромождать страницу. С учетом этого для визуализации отдельных элементов табличного юнита можно настроить пороговое число строк.

Все перечисленные выше программные механизмы существенно облегчают работу программиста, избавляя от монотонной, рутинной и однообразной работы, которая во многих других технологиях выполняется на сотнях и тысячах страниц.

Применение платформы MVE

Описываемая в статье технология изначально создавалась для разработки системы учета результатов научно-технической деятельности (РНТД). На основе платформы MVE был построен сайт регистрации РНТД, который используется для занесения в систему информации о публикациях (есть также поддержка работы над публикациями), научно-технических отчетах, объектах научно-технической собственности, конференциях, выставках, руководстве в дипломных и диссертационных работах. Доступ разграничивается по отдельным сотрудникам, подразделениям, институтам. В зависимости от прав доступа и должности сотрудника страницы претерпевают значительные изменения, что с легкостью обеспечивается технологией MVE. Данный сайт позволяет также получать ряд отчетов по научной деятельности, в частности, отчет по показателям результативности научной деятельности.

Другой сайт, созданный с использованием данной технологии, — сайт администрирования информационного пространства УрО РАН, реализующий работу с информацией о сотрудниках (в частности, конфигурирование прав доступа), подразделениях, институтах.

На основе платформы MVE создано еще два небольших сайта — специализированный сайт для совместной работы с документами, с опросами и тестированием, а также сайт поддержки аспирантуры.

Важная особенность — все эти сайты работают с одной базой данных и, как следствие, с одними информационными конструкциями. Юниты, разрабо-

танные для одного сайта, без изменений используются в других, избавляя от необходимости повторной разработки и демонстрируя высокую степень повторного использования кода.

Заключение

Рассмотренная в статье платформа разработки интернет-приложений может успешно применяться для создания сайтов, взаимодействующих с различными данными (причем не обязательно с базами данных).

Самые весомые плюсы технологии MVE — быстрота разработки и гибкость получаемого кода. Возможность малыми силами создавать большие сайты, имея при этом богатый потенциал по дальнейшей модернизации и повторному использованию кода.

Платформа MVE удобна для работы с большими объемами данных. Причин этому несколько. Во-первых, непосредственно взаимодействие с источником данных возложено на разработчика, значит продуктивность данного взаимодействия определяется его квалификацией и навыками. Во-вторых, в силу своей архитектуры и принципов работы MVE позволяет не экономить на отдельных деталях и выдавать такой объем и состав данных, который требуется. Этому способствует механизм стека юнитов. Там, где в иных концепциях разработчик вынужден включать блок HTML-кода в страницу, ужимая и комкая его, здесь можно легко и быстро создать отдельную страницу.

Автоматическая генерация страниц хотя и существенно ускоряет разработку, но накладывает некоторые ограничения на формируемые страницы, поэтому данная технология наилучшим образом подходит для сайтов с "шаблонными" страницами.

Эффективность практического применения платформы подтверждена серией сайтов, в основе которых лежит технология MVE. Следует также отметить, что в случае разработки нескольких сайтов, оперирующих одной базой, можно добиться очень хороших результатов, используя общие юниты.

После государственной регистрации планируется создание сайта проекта по адресу cris.uran.ru (исходный код MVE, документация, примеры).

Список литературы

1. **Фримен А., Сандерсон С.** ASP.NET MVC 3 Framework с примерами на C# для профессионалов. 3-е изд. СПб.: Вильямс, 2012. 672 с.

2. **Мак-Дональд М., Фримен А., Шпушта М.** ASP.NET 4 с примерами на C# 2010 для профессионалов. 4-е изд. СПб.: Вильямс, 2011. 1424 с.

3. **Мелансон Б., Нордин Д., Луиси Ж.** и др. Профессиональная разработка сайтов на Drupal 7. СПб.: Питер, 2013. 688 с.

4. **Ченцов П. А.** Платформа разработки Web-приложений MVE // Труды XX Всеросс. науч.-метод. конф. "Телематика 2013", г. Санкт-Петербург, 24—27 июня 2013 г. СПб.: Университетские телекоммуникации, 2013. Т. 2. С. 292—293.

P.A. Chentsov, Senior Researcher, e-mail: chentsov.p@uran.ru, Institute of mathematics and mechanics UrB RAS, Ekaterinburg

MVE Web-Development System

The development platform MVE (Model/View/Events) for web-applications is considered. It's based on the free software PHP, Apache, MySQL. The platform is designed for database applications, but it is possible to use MVE with any different data sources, like files system or electronic spreadsheets. The system is developed with object-oriented approach. One of the main features of MVE is the automatic generation of HTML pages. The process of developing sites reduces to the construction of the information structure object model, which automatically generates HTML pages. This approach provides a program compactness and high level of code reuse. Most expedient use of MVE is designing websites with template interface and working with complex data structures. MVE mechanisms allows to create and maintain sites a minimum number of site developers.

Keywords: *web-application, development system, automatic generation of pages*

References

1. **Frimen A., Sanderson S**. *ASP.NET MVC 3 Framework s primerami na C# dlja professionalov*. 3-e izd. SPb.: Vil'jams, 2012. 672 p.

2. **Mak-Donal'd M., Frimen A., Shpushta M.** *ASP.NET 4 s primerami na C# 2010 dlja professionalov*. 4-e izd. SPb.: Vil'jams, 2011. 1424 p.

3. **Melanson B., Nordin D., Luisi Zh.** i dr. *Professional'naja razrabotka sajtov na Drupal 7.* SPb.: Piter, 2013. 688 p.

4. **Chencov P. A.** Platforma razrabotki Web-prilozhenij MVE. *XX Vseross. nauch.-metod. konf. Telematika 2013,* Sankt-Peterburg, 24—27 ijunja 2013. SPb.: Universitetskie telekommunikacii. 2013. Vol. 2. P. 292—293.

УДК 004.414.23+004.415.5

А. О. Соколов, аспирант, инженер-конструктор, e-mail: wedmeed@mail.ru, Саратовский государственный технический университет им. Гагарина Ю. А., ОАО "КБ Электроприбор", г. Саратов

Оценка выполнимости наборов задач реального времени

Представлен тест, позволяющий оценить возможность выполнения (тест выполнимости) набора задач реального времени. В отличие от существующих тестов он предусматривает использование расширенной модели программного обеспечения, включающей регулярные задачи, задачи строгого реального времени и задачи с одинаковыми приоритетами. Предложен также ряд оптимизаций, позволяющих повысить применимость теста и используемой модели. Полученные выводы подтверждены результатами моделирования на большом числе наборов задач со случайными параметрами.

Ключевые слова: *тест выполнимости, программное обеспечение, задачи, планирование, реальное время, фактор утилизации, время отклика*

Введение

При разработке программного обеспечения (ПО) систем реального времени (РВ) важным показателем качества является предсказуемость [1]. Для его повышения могут использоваться различные методики, зависящие от структуры ПО, от применяемых при разработке инструментов, от используемой модели.

В большинстве существующих моделей ПО систем РВ применяют разделение на задачи и запросы [2—12]. При таком подходе одним из важнейших инструментов для повышения предсказуемости является тест, позволяющий оценить возможности выполнения набора задач (тест выполнимости) в системе.

Тест выполнимости может исполняться как на стадии создания объектного кода ПО (если число и параметры задач постоянны), так и во время выполнения (для систем с динамическим созданием задач). Для систем РВ наиболее характерен первый вариант.

Тест выполнимости использует параметры задач, определяемые моделью. Соответственно, чем более гибкой является модель, тем сложнее тест выполнимости в силу увеличения множества возможных комбинаций. Это обстоятельство является существенным препятствием при попытке включить в систему задачи, обладающие определенной спецификой. По этой причине в большинстве работ [4, 7, 10—12, 14] используют модели, включающие только периодические и спорадические задачи "жесткого" (*hard*) РВ, не имеющие одинаковых приоритетов.

Если пользоваться моделью, представленной в работе [2], то базовые тесты выполнимости являются либо неприменимыми, либо требуют излишне строгих допущений. Этот факт приводит к увеличению уровня их пессимистичности (далее — пессимистичность, количество отрицательных результатов теста для фактически выполнимых наборов задач) в рамках этой модели. В настоящей работе рассматривается задача адаптации базового теста выполнимости [12] под указанную модель с минимальной пессимистичностью. Все обозначения в данной статье соответствуют обозначениям работы [2].

Основные принципы проведения теста выполнимости

Основным этапом теста выполнимости является определение фактора использования (*utilization factor*). Фактор использования показывает часть процессорного времени, которая необходима для выполнения заданного набора задач при худшем стечении обстоятельств [3, 15]:

$$U_{\{a,\ ...,\ b\}} = \sum_{i=a}^{b} C_i / T_i,$$

где $\{a, ..., b\}$ — номера задач, для которых рассчитывается фактор использования, $1 \leqslant a \leqslant n$, $1 \leqslant b \leqslant n$, n — общее число задач, C_i — максимальное время выполнения; T_i — период инициации.

Необходимым условием для выполнимости набора задач жесткого РВ является следующее:

$$U \leqslant 1. \qquad (1)$$

Основная часть базового теста выполнимости [4, 12] строится на понятии максимального времени отклика задачи R_i. Параметр R_i будет зависеть от максимального времени выполнения самой задачи τ_i и всех задач с приоритетом, выше или равным приоритету P_i этой

задачи. Для вычисления R_i используется следующая функция:

$$R_i = C_i + \sum_{\forall k} \lceil R_i / T_k \rceil C_k \qquad (2)$$

где k: $1 \leqslant k \leqslant n$, $k \neq i$, $P_k > P_i$.

Вычисление (2) проводится в итерационной процедуре. Первым приближением берется $R_i = C_i$. Процедура повторяется до тех пор, пока результат очередного вычисления R_i не будет равен предыдущему. Формула (2) соответствует базовой формуле из работы [4]. Ее можно использовать в системах жесткого реального времени для периодических и спорадических задач.

Выбор (2) в качестве основной формулы для расчета времени отклика обусловлен тем, что она основана на вычислении нагрузки на систему, создаваемой высокоприоритетными задачами. При этом если гарантируется, что параметры C_i и T_i рассматриваемых задач не изменяются и не могут быть нарушены в процессе выполнения, то характер и порядок возникновения запросов этих задач не влияет на максимальное время отклика. Таким образом, при добавлении нового типа задач в (2) достаточно ограничиться определением их максимальных периодичности и потребления ресурсов. Тест выполнимости считается успешным, если для всех задач в системе выполняется следующее неравенство:

$$R_i < D_i, \qquad (3)$$

где D_i — относительный крайний срок выполнения задачи.

Использование (2) в представленном виде невозможно в рамках применяемой модели, так как она обладает следующими специфическими свойствами:

— наличие задач с одинаковым приоритетом;

— наличие регулярных задач;

— наличие задач "строгого" (*firm*) РВ.

Для обозначения типа задач будем использовать параметр $type_i$, который определяет один из следующих типов: *ph* — периодическая задача жесткого РВ; *pf* — периодическая задача строгого РВ; *sh* — спорадическая задача жесткого РВ; *sf* — спорадическая задача строгого РВ; *rh* — регулярная задача жесткого РВ; *rf* — регулярная задача строгого РВ.

Учет задач с одинаковыми приоритетами

Возможны два способа влияния друг на друга задач с одинаковыми приоритетами. Если в системе, где выполняются всего две задачи τ_k и τ_l, одна из задач τ_k начинает выполняться процессором раньше другой τ_l, то время выполнения τ_l не влияет на время отклика τ_k.

Если рассматриваемая задача τ_k инициирована во время работы τ_l, то все оставшееся время работы τ_l необходимо добавить ко времени отклика τ_k. Если задачи инициированы одновременно, то время отклика отложенной задачи увеличится на время выполнения активированной.

Если в рассматриваемой системе появляется третья задача с приоритетом выше, чем у рассматриваемых, то выполнение последних в худшем случае (например, если третья задача спорадическая) может быть прервано в любой момент. При завершении запроса добавленной задачи, в общем случае неизвестно заранее, какая из задач (τ_k или τ_l), будет активирована, а какая вытеснена. Однако при любом стечении обстоятельств время вытеснения будет не больше времени выполнения активированной задачи.

Таким образом, при наихудшем стечении обстоятельств влияние одной задачи на другую равносильно влиянию задачи с теми же характеристиками, но более высоким приоритетом. Чаще всего заранее невозможно определить, какая задача будет выбрана на выполнение первой (большое число механизмов разрешения подобных ситуаций не позволяют сделать однозначного обобщения). В этом случае при расчете времени отклика любой задачи все задачи с тем же приоритетом должны быть рассмотрены как более высокоприоритетные.

С учетом отмеченного выше появляется возможность использовать (2) с небольшим дополнением, касающимся индексов k внутри суммы:

$$k: 1 \leqslant k \leqslant n,\ k \neq i,\ P_k \geqslant P_i. \qquad (4)$$

Учет регулярных задач

Учет регулярных задач отличается от учета периодических и спорадических задач. Это связано с тем обстоятельством, что за период времени T_i для регулярной задачи τ_i может быть инициировано более одного запроса. Такой случай характерен для ситуации, представленной на рис. 1. На рисунке изображены три последовательных запроса регулярной задачи. Моменты времени 0, t_1, t_3 и t_4 отмечают периоды задачи. Первый из изображенных запросов инициирован в конце первого периода в момент t_1, второй — в момент t_2, а третий — в самом начале третьего периода в момент t_3. Над запросами задач отмечен произвольный промежуток времени, равный периоду задачи. Как видно, в него попали все три запроса.

Если представить регулярную задачу как спорадическую, то минимальный период между инициацией запросов будет равен нулю (в соответствии с используемой моделью). Таким образом, расчет времени отклика для всех задач с меньшим приоритетом будет давать бесконечное значение.

В моделях, не предусматривающих регулярный тип задач, можно использовать решение на основе трансформации регулярной задачи в две спорадические задачи. При этом все четные запросы будут относиться к одной задаче, а нечетные — к другой задаче.

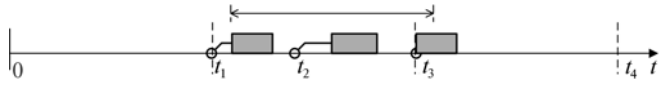


Рис. 1. Запросы регулярных задач

Приоритеты полученных спорадических задач не имеют значения для времени отклика низкоприоритетной задачи, кроме условия, что они у обеих задач высокие. Минимальным интервалом между инициацией запросов каждой из этих задач является T_i (следуя работе [2], $a_{i,j} \leqslant T_i j + \Phi_i$, $T_i(j+1) + \Phi_i \leqslant a_{i,j+2}$, что равносильно $a_{i,j+2} - a_{i,j} \geqslant T_i$, где $a_{i,j}$ — время инициации j-го запроса, Φ_i — фазовый сдвиг задачи, время инициации первого запроса). Таким образом, уравнение (2) с учетом (4) примет следующий вид:

$$R_i = C_i + \sum_{\forall k} \lceil R_i / T_k \rceil C_k + 2 \sum_{\forall l} \lceil R_i / T_l \rceil C_l, \qquad (5)$$

где k: $1 \leqslant k \leqslant n$, $k \neq i$, $P_k \geqslant P_i$, $type_k = ph$, pf, sh, sf, l: $1 \leqslant l \leqslant n$, $l \neq i$, $P_l \geqslant P_i$, $type_l = rh$, rf.

Условие (5) дает пессимистичный результат при оценке отклика для задач, время выполнения которых больше периода некоторой регулярной задачи с более высоким приоритетом. Если в определенный момент времени инициация запросов $\tau_{i,j}$ и $\tau_{i,j+2}$ произошла с разницей T_i, то запрос $\tau_{i,j+4}$ может произойти минимум через $2T_i$. Такая ситуация представлена на рис. 2. Инициация первого и третьего запросов произошла в моменты t_1 и t_2, что соответствует T_i, пятая инициация может произойти только в момент t_3, т. е. через $2T_i$ после t_2. Описанное преимущество можно использовать, если точно определить число инициаций запросов регулярной задачи за произвольный промежуток времени.

Пусть υ — некоторое число произошедших запросов регулярной задачи τ_i. Следуя работе [2], $T_i(j + \upsilon - 1) + \Phi_i \leqslant a_{i,j+\upsilon}$, $a_{i,j} \leqslant T_i j + \Phi_i$. Таким образом, υ произвольных инициаций запросов могут быть инициированы за минимальное время, определенное из следующего условия:

$$a_{i,j+\upsilon} - a_{i,j} \geqslant (T_i(j + \upsilon - 1) + \Phi_i) - (T_i j + \Phi_i) = 1(\upsilon - 1).$$

Для инициации $\upsilon - 1$ запросов спорадической задачи потребуется минимум то же самое время:

$$a_{i,j+\upsilon-1} - a_{i,j} \geqslant T_i(\upsilon - 1).$$

Следовательно, за одно и то же время число максимально возможных инициаций регулярной задачи на одно больше, чем число максимально возможных инициаций спорадической задачи с теми же параметрами. Учитывая этот факт, формулу (5) можно переписать в следующем виде:

$$R_i = C_i + \sum_{\forall k} \lceil R_i / T_k \rceil C_k + \sum_{\forall l} \lceil R_i / T_l + 1 \rceil C_l, \qquad (6)$$

где k: $1 \leqslant k \leqslant n$, $k \neq i$, $P_k \geqslant P_i$, $type_k = ph$, pf, sh, sf, l: $1 \leqslant l \leqslant n$, $l \neq i$, $P_l \geqslant P_i$, $type_l = rh$, rf.

Результат (6) всегда меньше или равен результату (5).

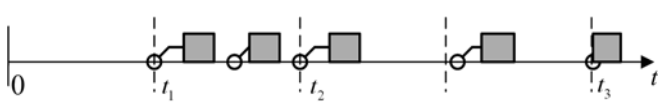


Рис. 2. Запросы регулярных задач на большом промежутке времени

Учет задач строгого реального времени

Все задачи строгого РВ в тесте выполнимости учитываются также, как и задачи жесткого РВ. Это обусловлено с тем, что наибольшая нагрузка на вычислительную систему с их стороны осуществляется, когда все запросы завершаются до инициации новых. При этом события инициации не пропускаются. Такое поведение соответствует поведению задач жесткого РВ.

Для подтверждения можно рассмотреть следующую ситуацию. Пусть имеется спорадическая задача строгого РВ τ_i (рис. 3). Выполнение запроса τ_i, инициированного в момент времени 0, было отложено по некоторым причинам. В момент t_1, когда запрос выполнялся, с момента инициации уже прошло минимальное время между инициациями (отмечено на рис. 3 стрелками над изображением запросов). В момент времени t_2 возникло новое событие инициации, но оно было пропущено, так как выполнение предыдущего запроса еще не завершено. После завершения первого запроса следующий мог быть инициирован только после момента времени t_3, так как в соответствии с моделью только в этот момент истекает период T_i после t_2.

Если рассмотреть рис. 3 с момента 0 до момента t_4 и опустить необработанное событие, то в данный промежуток времени задача фактически имела минимальное время между запросами, равное $t_3 - 0 > 2T_i$. Другими словами, если задача пропускает инициацию очередного запроса, то минимальное время до инициации следующего запроса увеличивается минимум на T_i. Так как при увеличении T_i результат (6) для задач с меньшим приоритетом, чем у рассматриваемой, уменьшается, то влияние задач строгого РВ является меньшим, чем влияние задач жесткого РВ. Такая ситуация имеет место для любых типов задач.

С учетом того, что ослабление влияния невозможно предсказать заранее, в тесте выполнимости учитывается наихудший результат (пропусков событий инициации не было, задача ведет себя как задача жесткого РВ). Таким образом, (5) можно применять для задач строгого РВ без каких-либо изменений.

Единственным существенным отличием является то, что для задач строгого РВ необязательно соблюдение условия (1). Таким образом, его проверку можно исключить из теста выполнимости.

Учет взаимного влияния периодических задач

Особенность периодических задач заключается в возможности полного предсказания временных границ их работы. В некоторых наборах задач возможны такие ситуации, когда возможные интервалы выполнения некоторых периодических задач никогда не будут накладываться друг на друга. Обзор работ с подоб-

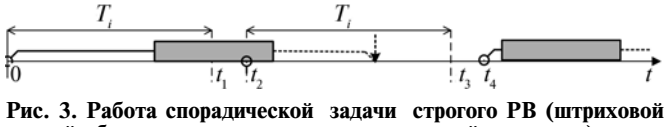


Рис. 3. Работа спорадической задачи строгого РВ (штриховой линией обозначено время до наступления крайнего срока)

ным решением представлен в работе [6], а в работах [13, 14] описанное упрощение используется для увеличения применимости тестов выполнимости. В данной работе оно также может быть использовано, однако требует доработки для возможности применения с рассматриваемыми типами задач.

Для анализа необходимо рассматривать все возможные случаи наложения периодических задач. Для этого достаточно рассмотреть промежуток времени, называемый гиперпериодом H:

$$Y = \mathrm{lcm}\left(\bigcup_{\forall i} T_i\right).$$

Гиперпериод является наименьшим общим кратным (lcm) периодов всех периодических задач. Взаимное расположение во времени запросов периодических задач с максимальным временем отклика будет повторяться во все последовательные промежутки времени, равные H и начинающиеся после максимального фазового сдвига Φ [3]. Таким образом, анализ можно ограничить рассмотрением любого промежутка времени, равного H и начавшегося позже Φ. Максимальный период рассчитывается следующим образом:

$$\Phi = \max\left(\bigcup_{\forall i} \Phi_i\right).$$

Если за время H ни разу не было момента, когда обе задачи были одновременно не в состоянии ожидания, то можно сделать вывод о том, что они взаимно не влияют друг на друга. При анализе времени отклика (6) задачи, не влияющие на рассматриваемую, могут не учитываться. Благодаря этому обстоятельству, за счет увеличения точности расчета времени отклика уменьшается пессимистичность теста выполнимости. Достаточное для этого условие может быть выражено в следующей форме:

$$\forall(t\colon \Phi < t < H + \Phi),$$
$$\hat{\tau}_k(t) \in \{ready, run\} \rightarrow \hat{\tau}_l(t) = wait, \qquad (7)$$

где k и l — порядковые номера задач, $1 \leqslant k \leqslant n$, $1 \leqslant l \leqslant n$, $k \neq l$, $type_k \in \{ph, pf\}$, $type_l \in \{ph, pf\}$, $\hat{\tau}_i$ — состояние задачи, $\hat{\tau}_i \in \{wait, run, ready, fail\}$ [2].

Проверка условия (7) требует полного анализа временного промежутка $\Phi < t < H + \Phi$. Для снижения трудоемкости анализ можно сократить. Для этого необходимо привязаться к моментам запуска и завершения запросов задач τ_k и τ_l (между этими точками проверка (7) будет давать один и тот же результат). В таком случае условие влияния задач друг на друга (т. е. обратное к (7)) может быть записано следующим образом:

$$\forall(g\colon \Phi_k + bT_k \leqslant s_{k,g} < \Phi_k + bT_k + H),$$

$$\exists\left(h : s_{k,g} - T_l < s_{l,h},\ f_{k,g} > s_{l,h},\ \begin{bmatrix} s_{l,h} \leqslant s_{k,g} < f_{l,h} \\ s_{l,h} < f_{k,g} \leqslant f_{l,h} \\ s_{k,g} \leqslant s_{l,h} < f_{k,g} \end{bmatrix}\right) \qquad (8)$$

где g и h — индексы запросов исследуемых задач, являющиеся натуральными числами, $1 \leqslant g$, $1 \leqslant h$; $s_{i,j}$ — момент начала выполнения запроса $\tau_{i,j}$; $f_{i,j}$ — момент окончания выполнения запроса $\tau_{i,j}$; b — некоторое натуральное число, такое, что

$$0 \leqslant b,\ \Phi \leqslant bT_k + \Phi_k. \qquad (9)$$

Исследуемый временной промежуток в (8) выбран так, чтобы периоды задачи τ_k, происходящие за этот промежуток, были полными. При этом проверяются коллизии только с теми запросами задачи τ_l, которые накладываются на период рассматриваемого запроса задачи τ_k. Наихудшей с точки зрения влияния на другие задачи оценкой для $s_{i,j}$ является момент инициации запроса $a_{i,j}$ ($a_{i,j} \leqslant s_{i,j}$), а для $f_{i,j}$ — максимально возможный момент отклика $a_{i,j} + R_i$ ($f_{i,j} \leqslant a_{i,j} + R_i$). Они могут быть оценены через параметры задачи:

$$\begin{aligned} \mathrm{bad}(s_{i,j}) &= a_{i,j} = \Phi_i + (j-1)T_i, \\ \mathrm{bad}(f_{i,j}) &= a_{i,j} + R_i = \Phi_i + (j-1)T_i + R_i. \end{aligned} \qquad (10)$$

где bad — функция, показывающая значение параметра при худшем, с точки зрения выполнимости сценарии ($\mathrm{bad}(s_{i,j})$ показывает самый ранний возможный момент начала выполнения запроса; $\mathrm{bad}(f_{i,j})$ — самый поздний возможный момент завершения запроса).

С учетом (10) и после упрощения условие (8) примет следующий вид:

$$\forall(g\colon b + 1 \leqslant g < b + 1 + H/T_k), \qquad (11)$$

$$\exists(h\colon \Phi_k - \Phi_l + (g-1)T_k + T_l - R_l < hT_l < \Phi_k - \Phi_l + (g-1)T_k + T_l + R_k). \qquad (12)$$

Таким образом, если при рассмотрении двух периодических задач τ_l и τ_k, для произвольного b, удовлетворяющего условию (9), при переборе всех натуральных g, ограниченных (11), существует натуральное h, удовлетворяющее (12), то тогда и только тогда τ_l и τ_k влияют друг на друга. При этом любое найденное h будет соответствовать номеру запроса задачи τ_l, выполнение которого потенциально может произойти в то же время, что и выполнение g-го запроса τ_k.

Для формул (8)—(12) индексы k и l определены в условии (7). Они описывают подмножество периодических задач из общего набора. Однако для сокращения трудоемкости из анализа можно исключить задачи типа *pf*. Для задач строгого времени возможна ситуация, когда $T_i < R_i$. В этом случае проверка (11) и (12) всегда будет давать положительный результат. В противном случае, если $T_i \geqslant R_i$, то задача становится фактически задачей жесткого РВ (и должна иметь тип *ph*).

При прогнозировании взаимного влияния задач в формуле (11) используются крайние сроки задач. Эти параметры невозможно точно рассчитать до проведения анализа взаимовлияния. Таким образом, расчет времени отклика для периодических задач должен проводиться в приведенной ниже итерационной процедуре.

Шаг 1. Рассчитывается время отклика каждой периодической задачи без учета влияния других периодических задач.

Шаг 2. Проводится анализ влияния всех неучтенных задач на исследуемую.

Шаг 3. Если появились влияющие задачи, которые не были учтены при расчете времени отклика, то выполняются последующие шаги, иначе расчет завершается.

Шаг 4. Рассчитывается уточненное время отклика с учетом всех периодических задач, чье влияние установлено.

Шаг 5. Выполняется переход к шагу 2.

При учете взаимного влияния периодических задач пессимистичность теста выполнимости не устраняется полностью. Это, как минимум, связано с тем обстоятельством, что если периодические запросы накладываются друг на друга, например, один раз за гиперпериод, то будет увеличено время отклика всех запросов. Однако дальнейшая оптимизация алгоритма связана с составлением расписаний, что значительно увеличивает сложность. Сложность алгоритма в представленном виде уже соответствует классу NP в силу двух взаимозависимых итерационных процедур, но может быть сокращена до условно полиномиальной (если останавливать алгоритм при выполнении условия (3) и опускать повторную проверку взаимовлияния).

Сравнение результатов тестов выполнимости

Прямой метод оценки пессимистичности теста выполнимости заключается в сравнении оценки теста с реальной выполнимостью для всех возможных наборов задач. При этом параметры задач должны удовлетворять следующим условиям:

$$\begin{cases} type_i \in \{ph, pf, sh, sf, rh, rf\}; \\ C_i < T_i; \\ D_i \geqslant C_i; \\ \forall(i: type_i \in \{ph, rh, sh\}),\ D_i \leqslant T_i; \\ P_i \leqslant n. \end{cases} \quad (13)$$

В соответствии с условиями (13) возможно создать неограниченное число наборов задач с параметрами $\{C_i, D_i, \Phi_i, T_i, P_i, type_i | i = \overline{1, n}\}$, так как некоторые параметры не имеют верхних ограничений. Таким образом, полная оценка не является осуществимой. Для проведения приближенной оценки можно ограничиться некоторыми значениями, встречающимися в реальных реализациях. В данной работе приняты следующие ограничения:

$$\begin{cases} 100 \leqslant T_i \leqslant 1000; \\ T_i : 100; \\ \Phi_i \leqslant 5T_i; \\ \forall(i: type_i \in \{pf, rf, sf\}),\ D_i \leqslant 5T_i; \\ n = 2, \ldots, 6. \end{cases} \quad (14)$$

В условиях (14) числовые параметры выбраны из следующих соображений:

- большое разрешение T_i позволяет анализировать наборы задач, ограниченные минимальным фактором использования $U = n\min(C_i)/\max(T_i) = n/1000$ (при уменьшении разрешения T_i минимальный фактор будет увеличиваться);
- T_i кратно 100, так как это позволяет сократить поиск наименьшего общего кратного;
- Φ_i и D_i выбраны на основе исследования реальных систем;
- диапазон n выбран из соображений оптимальности соотношения наглядности результатов и времени проведения тестов.

Условие (14) делает оценку возможной, однако на практике вычисление теста для всех полученных комбинаций займет слишком много времени. Для дополнительной оптимизации приняты следующие решения:

- в соответствии с методом Монте-Карло [16] для теста выбрано ограниченное число наборов (в данной работе — 1 000 000), каждый параметр которых выбирается случайно из возможных значений, ограниченных (13) и (14) (таким образом можно сказать, что набор выбран случайно из допустимого множества);
- сравнение результатов исследуемого теста выполнимости проводится не с результатами полного моделирования, а с результатами вычисления фактора использования, который сам является косвенной оценкой выполнимости набора задач [3];
- для наглядности и точности проводится сравнение с результатами базового теста выполнимости (одинаковые приоритеты отсутствуют, регулярные задачи рассматриваются как две спорадические задачи, крайний срок для задач строгого РВ вычисляется как $\min(T_i, D_i)$, анализ взаимовлияния периодических задач не проводится), пессимистичность которого в некоторой мере уже оценена в работах [3, 4, 11, 12];
- учет задач с одинаковым приоритетом вносит пессимизм, когда все остальные решения, приведенные в данной работе, его уменьшают; поэтому для наглядности предлагаемый тест выполнимости проводится для двух наборов задач с одинаковыми параметрами, отличие заключается только в том, что в первом наборе есть задачи с одинаковыми приоритетами, а во втором наборе приоритеты перераспределяются так, чтобы каждая задача имела оригинальный приоритет.

На рис. 4 представлены зависимости, рассчитанные при $n = 5$ для базового и предлагаемого тестов, удовлетворяющие следующему условию:

$$f(U) = Ns(U)/Nu(U),$$

где Nu — общее число наборов задач; Ns — число наборов задач, для которых тест выполнимости дал положительный результат.

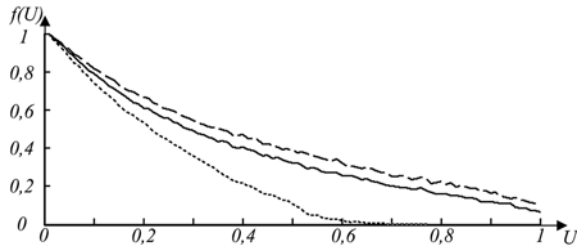


Рис. 4. Отношение положительных результатов тестов к общему числу тестов в зависимости от фактора использования наборов задач с $n = 5$ (сплошной линией обозначены результаты предлагаемого теста, штриховой линией — результаты предлагаемого текста без задач с одинаковыми приоритетами, пунктирной линией — результаты базового теста)

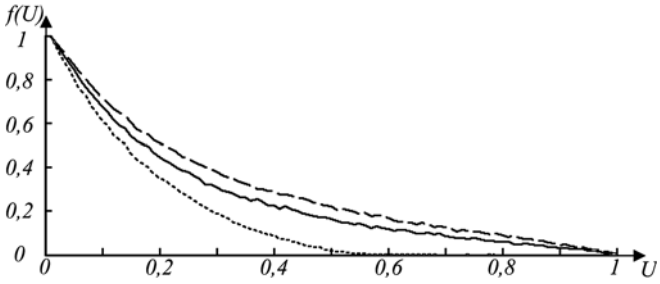


Рис. 5. Отношение положительных результатов тестов к общему числу тестов в зависимости от фактора использования наборов задач жесткого РВ с $n = 5$ (сплошной линией обозначены результаты предлагаемого теста, штриховой линий — результаты предлагаемого текста без задач с одинаковыми приоритетами, пунктирной линией — результаты базового теста)

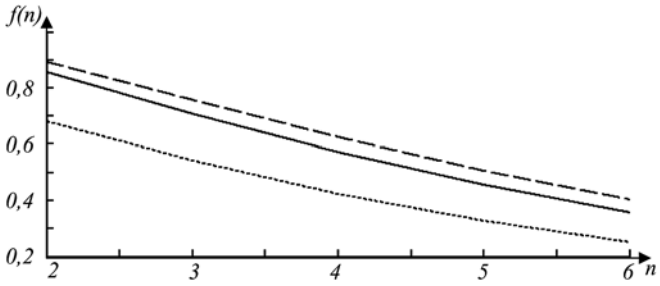


Рис. 6. Отношение положительных результатов тестов к общему числу тестов в зависимости от числа задач в наборе n (сплошной линией обозначены результаты предлагаемого теста, штриховой линий — результаты предлагаемого теста без задач с одинаковыми приоритетами, пунктирной линией — результаты базового теста)

Эти зависимости позволяют оценить влияние "сложности" набора задач, выраженной в факторе использования, на результаты теста.

Как видно на рис. 4, существует много наборов задач, для которых тест выполнимости дает положительный результат при факторе использования $U = 1$. Такая ситуация возможна, когда в наборе есть низкоприоритетные задачи строгого РВ, при исключении которых фактор использования становится меньше единицы.

Для сравнения на рис. 5 приведены результаты тестов при отсутствии в них задач строгого РВ.

На рис. 6 представлена зависимость того же самого отношения, но от числа задач в наборе:

$f(n) = Ns(n)/Nu(n)$.

Заключение

Как видно из представленных выше результатов, модель, описанная в работе [2], наряду с расширением гибкости и применимости способна уменьшить пессимистичность тестов выполнимости. Негативными эффектами от применения данной модели является увеличение вычислительной сложности теста выполнимости, что ограничивает его применение в процессе эксплуатации.

Таким образом, представленная в настоящей работе модель больше подходит для систем РВ со статически определенными задачами. Изменение состава ПО в процессе работы в таких системах возможно только путем отключения некоторых функций из изначально описанного набора. При этом, с точки зрения операционной системы, диспетчера или любого другого модуля, поддерживающего механизм задач и запросов, задачи не исключаются из системы, а просто при каждой очередной инициации сразу генерируют событие завершения. Такая ситуации довольно часто встречается на практике, например в цифровых регуляторах [2].

Представленный тест выполнимости позволяет с малой пессимистичностью гарантировать выполнимость заданного набора задач. Кроме обеспечения показателя предсказуемости ПО использование представленной методики позволяет точнее определить производительность по сравнению с применением базового теста. Более емкие в плане потребления процессорного времени наборы задач могут быть выполнены на менее мощных вычислительных платформах.

Дальнейшее снижение пессимистичности теста выполнимости возможно путем изменения алгоритма проверки взаимовлияния периодических задач и путем введения в модель новых ограничений. Первое решение увеличит вычислительную сложность теста, а второе решение может снизить гибкость. Более приемлемым подходом может оказаться разработка новых моделей ПО, однако для этого необходимо провести глубокий анализ существующих систем и принципов их функционирования.

Предлагаемый автором тест выполнимости может быть использован в исследованиях, направленных на разработку принципов планирования и модификацию архитектур операционных систем для систем РВ с применением расширенной модели ПО [2]. Его применение также возможно и при использовании существующих инструментальных средств, для которых характерны описанные в данной работе допущения.

Список литературы

1. **Майерс Г. Д.** Надежность программного обеспечения. М.: Мир, 1980.

2. **Соколов А. О.** Модель программного обеспечения систем реального времени // Программная инженерия. 2014. № 7. С. 9—16.

3. **Buttazzo G. C.** Hard real-time computing systems: predictable scheduling algorithms and applications. New York: Springer, 2011.

4. **Tindell K. W., Burns A., Wellings A. J.** An extendible approach for analyzing fixed priority hard real-time tasks // Real-Time Systems. 1994. Vol. 6, N. 2. P. 133—151.

5. **Кавалеров М. В.** Современное состояние исследований и практических внедрений, связанных с проблемами планирования задач реального времени в системах управления, контроля и измерения // Труды Третьей российской конференции с международным участием "Технические и программные средства

систем управления, контроля и измерения". М.: ИПУ РАН, 2012. С. 1360—1372.

6. **Sha L., Abdelzaher T., Årzén Karl-Erik** et al. Real time scheduling theory: A historical perspective // Real-time systems. 2004. Vol. 28, N. 2—3. P. 101—155.

7. **Кавалеров М. В.** Планирование задач в системах автоматизации и управления при нестандартных ограничениях реального времени: дис. ... канд. техн. наук. Пермь, 2007. 143 с.

8. **Таненбаум Э.** Современные операционные системы. 3-е изд. СПб.: Питер, 2010. 1120 с.

9. **Isovic D., Fohler G.** Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints // Proc. of Real-Time Systems Symposium, 2000. The 21st IEEE. Orlando: IEEE, 2000. P. 207—216.

10. **Isovic D., Fohler G.** Handling mixed sets of tasks in combined offline and online scheduled real-time systems // Real-time systems. 2009. Vol. 43, N. 3. P. 296—325.

11. **Wang W., Mok A. K., Fohler G.** Generalized pre-scheduler // Proc. of Real-Time Systems, 2004. ECRTS 2004. 16th Euromicro Conference. Catania: IEEE, 2004. P. 127—134.

12. **Joseph M., Pandya P.** Finding response times in a real-time system // The Computer Journal. 1986. Vol. 29, N. 5. P. 390—395.

13. **Palencia J.C., Gonzalez H. M.** Schedulability analysis for tasks with static and dynamic offsets // Proc. of Real-Time Systems Symposium, 1998. The 19th IEEE. Madrid: IEEE, 1998. P. 26—37.

14. **Fohler G.** Predictably Flexible Real-Time Scheduling // Advances in Real-Time Systems. Berlin: Springer Berlin Heidelberg, 2012. P. 207—221.

15. **Marwedel P.** Embedded system design. Dordrecht: Springer, 2006. 241 p.

16. **Соболь И. М.** Численные методы Монте-Карло. М.: Наука, 1973. 312 с.

A. O. Sokolov, Postgraduate Student, Design Engineer, e-mail: wedmeed@mail.ru, Saratov State Technical University of Gagarin Yu. A., "KB Electropribor", Saratov

A Feasibility Test for Real-Time Systems Software

The paper presents a feasibility test for real-time tasks set. Unlike existing tests it use an advanced software model. This model includes regular tasks, firm real-time tasks and tasks with equal priorities.

Allowing defined features reduces pessimism of the test results. An important advantage is the set feasibility when a utilization factor is greater than 1. This is possible because firm real-time tasks benefits are introduced. Also in this paper several other optimizations are proposed. They improve applicability of the test.

Owing to the universality of the model used proposed test is compatible with older models. Sets of tasks created earlier can be checked without any changes.

Simulations with a large number of tasks sets with random parameters confirm results of this work. In all experiments pessimism of the proposed test is lower than pessimism of the base test.

Keywords: *feasibility test, software, tasks, scheduling, real-time, utilization factor, response time*

References

1. **Majers G. D.** *Nadezhnost' programmnogo obespechenija*. M.: Mir, 1980.

2. **Sokolov A. O.** Model' programmnogo obespechenija sistem real'nogo vremeni. *Programmnaja ingenerija*. 2014. N. 7. P. 9—16.

3. **Buttazzo G. C.** *Hard real-time computing systems: predictable scheduling algorithms and applications*. New York: Springer, 2011.

4. **Tindell K. W., Burns A., Wellings A. J.** An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*. 1994. Vol. 6, N. 2. P. 133—151.

5. **Kavalerov M. V.** Sovremennoe sostojanie issledovanij i prakticheskih vnedrenij, svjazannyh s problemami planirvoanija zadach real'nogo vremeni v sistemah upravlenija, kontrolja i izmerenija. *Trudy Tret'ej rossijskoj konferencii s mezhdunarodnym uchastiem "Tehnicheskie i programmnye sredstva sistem upravlenija, kontrolja i izmerenija"*. M.: IPU RAN, 2012. P. 1360—1372.

6. **Sha L., Abdelzaher T., Årzén Karl-Erik** et al.Real time scheduling theory: A historical perspective. *Real-time systems*. 2004. Vol. 28, N. 2—3. P. 101—155.

7. **Kavalerov M. V.** Planirovanie zadach v sistemah avtomatizacii i upravlenija pri nestandartnyh ogranichenijah real'nogo vremeni: dis. ... kand. tehn. nauk. Perm', 2007. 143 p.

8. **Tanenbaum Je.** *Sovremennye operacionnye sistemy*. 3-e izd. SPb.: Piter, 2010. 1120 p.

9. **Isovic D., Fohler G.** Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints. *Proc. of Real-Time Systems Symposium, 2000. The 21st IEEE*. Orlando: IEEE, 2000. P. 207—216.

10. **Isovic D., Fohler G.** Handling mixed sets of tasks in combined offline and online scheduled real-time systems. *Real-time systems*. 2009. Vol. 43, N. 3. P. 296—325.

11. **Wang W., Mok A. K., Fohler G.** Generalized pre-scheduler. *Proc. of Real-Time Systems*, 2004. *ECRTS 2004. 16th Euromicro Conference*. Catania: IEEE, 2004. P. 127—134.

12. **Joseph M., Pandya P.** Finding response times in a real-time system. *The Computer Journal*. 1986. Vol. 29, N. 5. P. 390—395.

13. **Palencia J.C., Gonzalez H. M.** Schedulability analysis for tasks with static and dynamic offsets. *Proc. of Real-Time Systems Symposium, 1998*. The 19th IEEE. Madrid: IEEE, 1998. P. 26—37.

14. **Fohler G.** Predictably Flexible Real-Time Scheduling. *Advances in Real-Time Systems*. Berlin: Springer Berlin Heidelberg, 2012. P. 207—221.

15. **Marwedel P.** *Embedded system design*. Dordrecht: Springer, 2006. 241 p.

16. **Sobol' I. M.** *Chislennye metody Monte-Karlo*. M.: Nauka, 1973. 312 p.

УДК 004.9

А.А. Бездушный, аспирант, e-mail: andrey.bezdushny@gmail.com,
Московский физико-технический институт, г. Долгопрудный

Концептуальные положения и архитектура системы семантического управления личной информацией

Рассмотрена задача управления личной информацией, предложена архитектура системы семантического управления личной информацией, а также представлен прототип системы, реализованный в соответствии с этой архитектурой. Предлагаемая методика развивает идею подхода Semantic Desktop — способа организации информационного пространства человека в соответствии со стандартами Semantic Web и Linked Open Data.

Ключевые слова: *управление информацией, управление знаниями, Semantic Desktop, Semantic Web, Linked Open Data, анализ данных, категоризация*

Введение

Для решениия вопросов эффективной организации и работы с данными, а также работы со знаниями предназначены системы управления личной информацией (*Personal Information Management Systems*). Одним из первых свое видение подобной системы в 1945 г. высказал Вэнивар Буш в эссе "Как мы можем мыслить" [1]. В. Буш описал устройство под названием Мемекс (*Memex*), в котором люди могут хранить всю свою личную информацию — мысли, записи, книги и которое может выдавать нужную информацию с достаточной скоростью и гибкостью. В основу работы Мемекс В. Буш закладывал механизмы ассоциативных ссылок и примечаний. Устройство, по его замыслу, должно было точно имитировать ассоциативные процессы человеческого мышления, исключая присущие человеку недостатки, такие как забывание информации. Одной из технологий, необходимых для реализации своего устройства, В. Буш считал возможность организации хранилища, содержащего такое, практически неограниченное количество информации, что "даже если бы пользователь вставлял в него по 5000 страниц сведений в день, ему бы потребовались сотни лет, чтобы заполнить свое хранилище". Сейчас стоимость жестких дисков мала настолько, что человек может хранить всю изученную им информацию в течение неограниченного количества времени, увеличивая при необходимости объем хранилища путем добавления нового жесткого диска. Таким образом, на пути создания Мемекса в настоящее время стоят задачи, связанные с проектированием гибкой системы, способной помогать человеку при выполнении ежедневных задач, дополняя и структурируя его мыслительные процессы.

Управление знаниями

Перед тем как переходить к рассмотрению вопросов управления информацией, необходимо определить основные понятия, такие как данные, информация и знания. В разных работах [2—4] приводятся различные определения этих понятий и точки зрения на их взаимосвязь. Можно, однако, выделить общую идею, в соответствии с которой эти понятия последовательно порождаются друг из друга. Данные — это "сырые" сведения, например, набор символов или последовательность байт. Информация — это данные, к которым добавлена семантика, например, утверждения или документы. Знания — это закономерности (принципы, связи, законы), сформированные в ходе практической деятельности или анализа информации. В ряде работ знания разделяют на явные и неявные. Неявные знания — это знания, не выраженные в формальном виде, они чаще обладают характеристиками, связанными с конкретным человеком — определяют его навыки или умения, накопленные с опытом. Явные знания — знания, сформулированные и систематизированные в виде, пригодном для обмена.

Выделяют два направления исследований в работе со знаниями — управление знаниями (*Knowledge Management*) и управление личными знаниями (*Personal Knowledge Management*). В исследованиях по управлению знаниями в основном изучают процессы, возникающие в ходе создания и обмена знаниями в рамках коллективов, организаций и других более крупных структур. В управлении личными знаниями бóльшее внимание уделяется вопросам организации, поиска и извлечения знаний, накопленных конкретным человеком.

Управление информацией

Управление личной информацией (*Personal Information Management*) изучает вопросы, во многом схожие с изложенными в предыдущем разделе, основное отличие состоит в области изучаемых знаний. Управление личной информацией фокусируется на управлении сведениями, с которыми ежедневно работает человек, т. е. со знаниями, выраженными в формализованном виде, — документами, письмами и

пр., тогда как управление личными знаниями в большей степени сосредоточено на неявных знаниях и подходах к их преобразованию в явные знания.

С каждым годом количество информации, с которой ежедневно сталкивается человек, растет. Все больший ее объем переходит в электронный формат — публикуется в сети или сохраняется на персональных компьютерах. Эта информация распределяется между различными источниками, в которых она хранится в разнородных форматах. Часть сведений может храниться в виде документов, другая — в виде ссылок или заметок, третья — в контексте не связанных между собой информационных систем. Такая организация данных приводит к *фрагментации информации* — в рамках различных источников человеку приходится поддерживать различные, зачастую не связанные между собой, но обладающие общей структурой организационные схемы. Разнородность форматов хранения данных затрудняет процесс задания зависимостей между этими схемами. В результате сведения о таких взаимосвязях фиксируются только в памяти человека. Со временем эти сведения неизбежно забываются, затрудняя процесс воссоздания контекста работы и поиска ресурсов, которые использовались ранее.

Системы управления личной информацией автоматизируют процессы ведения и работы с *информационным пространством* — совокупностью всех сведений, с которыми человек работает в настоящее время или работал ранее. Рассмотрим основные функции, которые, с точки зрения автора, должны выполнять системы управления личной информацией.

1. Ведение информационного пространства и структуризация находящихся в нем *информационных ресурсов*. Под информационными ресурсами будем понимать любые данные, имеющие важность для человека и выделяемые им в отдельную сущность. Это могут быть файлы, заметки, посещенные веб-страницы, письма и пр. Системой управления личной информацией должны обеспечиваться процессы, способствующие формализации информационных ресурсов, а также операций, выполняемых над ними.

2. Поиск по информационному пространству. Часто человек сталкивается с задачей повторного поиска информации, с которой он работал ранее. В таком случае при поиске он обычно обладает большим количеством косвенной информации, касающейся искомого ресурса. Эта информация, как правило, имеет ассоциативный формат, например, с какими еще ресурсами велась работа одновременно с искомым, в какой период времени это выполнялось, какая последовательность действий была совершена при первичном нахождении ресурса. Системы управления личной информацией должны поддерживать ведение такого рода метаданных и предоставлять возможность поиска по ним.

3. Автоматический анализ информационного пространства. Поскольку информационные ресурсы хранятся в структурированном виде, становится возможным проведение их автоматической обработки и анализа. Можно выделить несколько аспектов такого анализа. Один из них — автоматическое пополнение метаданных об информационных ресурсах сведениями, найденными в сети. Другим аспектом является поиск схожих или связанных ресурсов как в рамках информационного пространства пользователя, так во внешней сети.

4. Еще одним процессом, который может быть автоматизирован системами управления личной информацией, является категоризация ресурсов информационного пространства. Часть ресурсов может быть категоризирована системой автоматически, например, научные публикации, музыка, видеоматериалы и прочие ресурсы, категории для которых определены в сети. Другая часть — с участием человека, в таком случае частичное распределение по категориям проводится пользователем, а системой на основании этого распределения предлагаются категории для новых ресурсов.

5. Возможность совместной работы. При ведении общего проекта люди часто сталкиваются с необходимостью совместной работы над частью информации. Для такого случая системы управления личной информацией должны поддерживать обмен информационными ресурсами, их метаданными или частичное объединение информационных пространств разных пользователей системы, а также предоставлять возможности для комментирования и обсуждения информационных ресурсов.

Таким образом, систему управления личной информацией можно рассматривать как своеобразного интеллектуального цифрового помощника, сопровождающего и помогающего пользователю вести его информационное пространство. Дополняя сырые данные структурой и семантикой, пользователь получает возможность автоматизировать выполняемые им интеллектуальные процессы.

Существующие подходы к управлению информацией

Несмотря на существование большого числа работ в области управления личной информацией, в настоящее время распространены лишь так называемые "персональные органайзеры", рассматривающие самые простые задачи, такие как планирование событий, установка напоминаний, ведение заметок и контактов, работа с электронной почтой. Наиболее популярными примерами таких органайзеров являются Microsoft Outlook, Mozilla Thunderbird и др. Предоставляя возможности для хранения ограниченных типов информации, эти средства тем не менее опускают вопросы управления накопленными сведениями — формирования взаимосвязей между данными, организации составных структур, совместной работы.

Рассмотрим основные направления работ, опубликованных в последние годы по теме управления личной информацией. В работе [5] описана концепция *Semantic Desktop* — подхода к организации данных на персональном компьютере, в соответствии с которым любая информация, используемая пользователем, включая файл, e-mail или событие календаря, рассматривается как RDF-ресурс с собственным уникальным идентификатором. В этой работе вводится модель личной информации (*Personal Information*

Model — PIMO), формализующая ментальную модель информационного пространства, составленную пользователем. Основная задача PIMO — предоставить общую модель данных, с которой смогут работать различные приложения, используемые пользователем. Единое представление данных предоставит больше возможностей для организации более гибкой интеграции между приложениями. К недостаткам описанной в работе [5] системы Gnowsis можно отнести тот факт, что основной акцент делается на организации модели данных и ее совместном использовании различными приложениями, в то время как вопросы управления накопленными данными почти не рассматриваются. Этим вопросам посвящены следующие работы: SemEx [6], IRIS [7], Haystack [8], MyLifeBits [9], DeepaMehta [10]. В системах SemEx, IRIS и Haystack данные представляются в иерархическом виде. В основе иерархии лежат наиболее распространенные типы данных, такие как e-mail, контакты, проекты. В системах IRIS и Haystack для каждого типа данных определен набор интерфейсов, предоставляющих базовые операции, такие как возможности ответа или пересылки письма, создания события или напоминания. В системе Haystack дополнительно предоставляется возможность настройки стандартных и создания собственных визуальных представлений данных, а также определяются программные интерфейсы для создания дополнительных операций над данными. В работах [9] и [10] рассмотрены альтернативы к иерархическому подходу представления данных. В работе [9] предложены интерфейсы отображения ресурсов с использованием временной шкалы. По мнению авторов работы [9], введение временной шкалы позволяет более наглядно отобразить ресурсы, а также увеличить число одновременно выводимых ресурсов. В системе DeepaMehta данные предоставляются пользователю в форме тематической карты (*topic map*) — ориентированного графа, узлами которого являются ресурсы, определенные пользователем.

В работе Beagle++ [11] подробно рассмотрены вопросы ранжирования ресурсов, полученных в результате поискового запроса. Ранжирование проводится на основании объединения результатов, полученных с помощью алгоритмов ObjectRank и TF/IDF.

Работы iMecho [12] и Feldspar [13] рассматривают вопрос использования ассоциаций при поиске ресурсов. В работе [12] предложено формировать журнал работы пользователя с ресурсами для дальнейшего анализа и выделения зависимостей между ресурсами. В системе Fledspar предоставляется удобный интерфейс для ассоциативной навигации по ресурсам, а также реализуется возможность осуществлять поиск ресурсов на основании информации, связанной с ними. В работе Desktop Gateway [14] помимо интеграции между приложениями также рассмотрены вопросы использования данных, полученных из сети.

Перечисленные работы делают больший упор на формирование информационного пространства и работу с ним, в меньшей мере затрагивая вопросы анализа данных и автоматизации действий, выполняемых пользователем. В них слабо освещены вопросы объединения информационных пространств различных пользователей и совместной работы с ними.

Предлагаемое решение

В данном разделе приведено описание архитектуры предлагаемого решения, а также проведено сравнение двух схем, соответствующих разным моделям работы пользователей с данными, одна из которых не использует систему управления информацией (рис. 1),

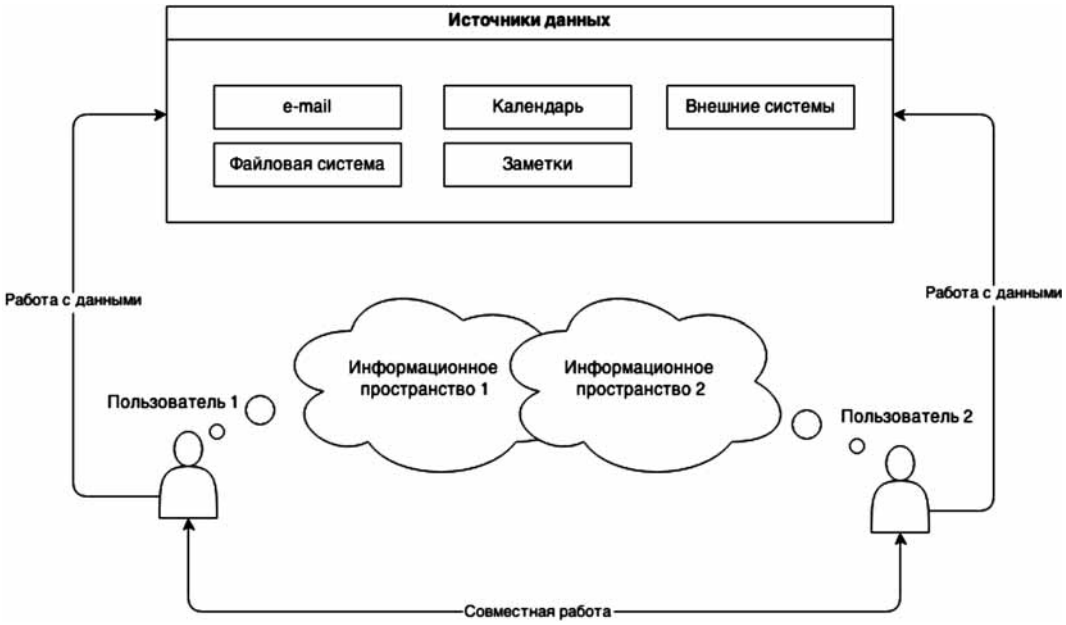


Рис. 1. Схема работы пользователей без использования системы

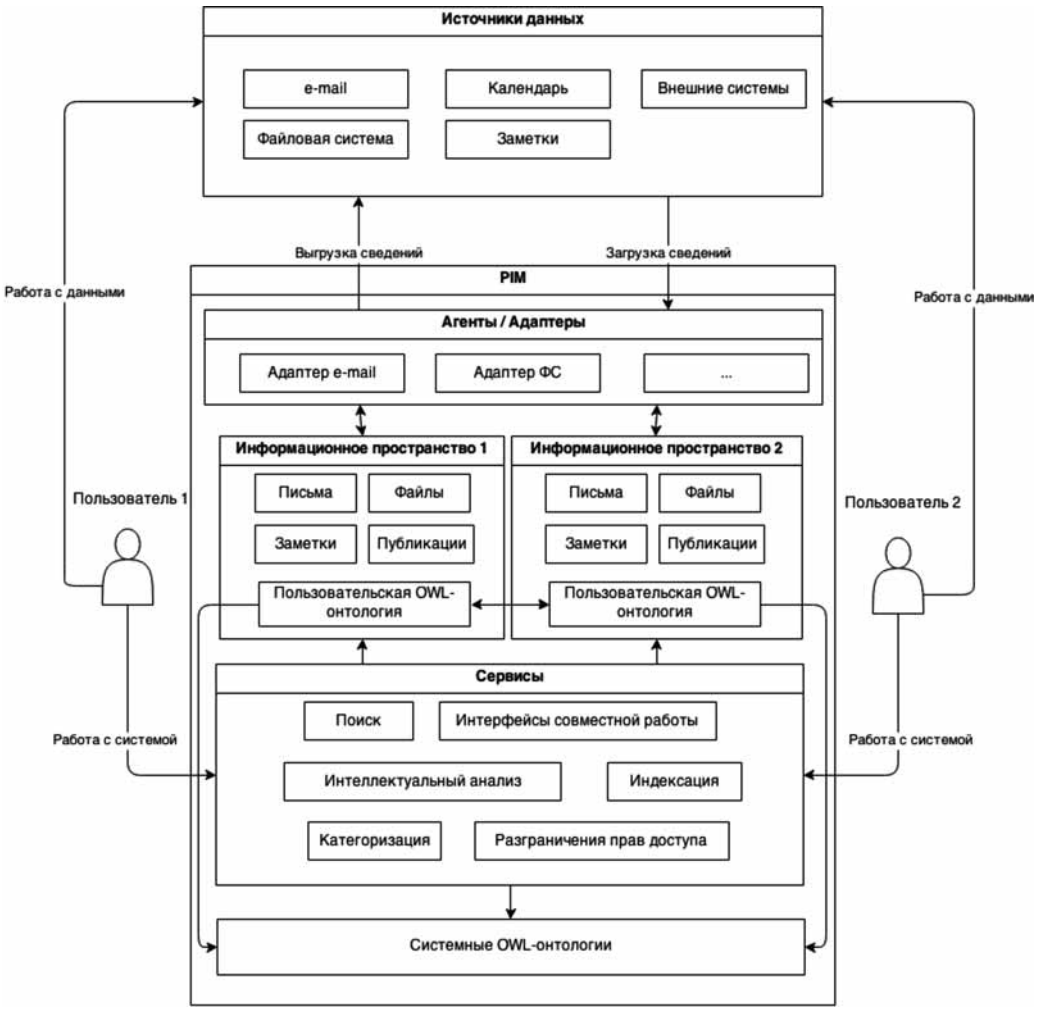


Рис. 2. Схема работы пользователей с использованием системы

а другая использует (рис. 2). На этих схемах представлены режимы работы двух пользователей, чьи информационные пространства частично пересекаются. В процессе работы каждый пользователь пополняет собственное информационное пространство, состоящее из разнородных данных — документов, событий, писем. Работа с ресурсами ведется с помощью различных приложений и информационных систем.

Без использования системы (см. рис. 1) информационные пространства пользователей нигде формально не определены, только сам пользователь может определить, какие данные связаны между собой и как именно. Вследствие этого все дальнейшие взаимодействия с данными — поиск, совместная работа, формирование иерархической структуры могут проводиться лишь в рамках того приложения, которое отвечает за конкретный тип ресурсов. Поскольку информация делится между различными приложениями, большое количество метаданных о ресурсах, таких как иерархическая организация, связи и зависимости между ресурсами, могут дублироваться в каждом из них.

Системы управления личной информацией (см. рис. 2) вводят дополнительный уровень организации данных, позволяют пользователю явно определить свое информационное пространство и предоставляют интерфейсы для работы с ним.

В основе предлагаемой архитектуры лежит использование OWL-онтологий. Язык веб-онтологий (OWL) [15] позволяет формально описать концептуальную модель предметной области. Конструкции, определяемые языком OWL, соответствуют понятиям дескриптивной логики, что позволяет проводить автоматический анализ описанных сведений.

В предлагаемой архитектуре *информационное пространство* пользователя представляет собой RDF-репозиторий, ресурсы которого соответствуют понятиям пользовательской OWL-онтологии. Для наполнения онтологии могут использоваться понятия, определенные в системных онтологиях (такие как "Проект", "Персона", "E-mail" и др.), или могут быть созданы новые понятия. Созданные пользователями ресурсы хранятся в единой RDF-базе данных (в прототипе, представленном далее, используется репозиторий Sesame).

Пополнение информационного пространства пользователя возможно двумя способами: пользователь может либо самостоятельно создавать ресурсы с помощью интерфейсов системы, либо настроить автоматический импорт ресурсов из внешних источников (таких как файловая система, почтовый сервер и т.п.). За последнее отвечает уровень *адаптеров к внешним источникам*. Адаптеры в фоновом режиме проводят синхронизацию ресурсов информационного пространства пользователя с данными, созданными во внешних источниках. Это предоставляет возможность пользователям работать с привычными приложениями.

Следующим уровнем рассматриваемой архитектуры является уровень сервисов, обеспечивающих базовые функциональные возможности системы. В рамках системы реализуются сервисы "Поиск", "Категоризация", "Индексация", "Анализ данных", "Совместная работа", "Разграничение прав доступа". Некоторые из них будут рассмотрены более детально.

Требования к системе

Сформулируем основные требования, которым должна соответствовать проектируемая система.

1. Хранение информации пользователя в формате RDF.

2. Предоставление системной OWL-онтологии, в которой определены наиболее часто используемые пользователями классы.

3. Возможность загрузки в информационное пространство пользователя данных, сформированных сторонними приложениями (e-mail, контакты, календари, файлы). В зависимости от типа загружаемые данные должны представляться в системе RDF-ресурсами различных классов из OWL-онтологии.

4. Автоматическое формирование метаданных на основе анализа загруженных ресурсов, таких как отправитель и получатель электронного письма, если было загружено e-mail-сообщение, автор и название статьи, если был загружен файл, представляющий собой научную статью.

5. Автоматизированное определение связей между загруженными в систему ресурсами.

6. Автоматизированная категоризация загруженных ресурсов — на основании содержимого, если загружен произвольный файл, или на основании общих свойств, если файл соответствует конкретному классу ресурсов, определенному в OWL-онтологии.

7. Поиск новых, потенциально интересных пользователю ресурсов в сети (в частности, в репозиториях Linked Open Data) на основании уже имеющихся в системе ресурсов.

8. Интерфейс системы должен быть прост, интуитивно понятен, больший упор следует делать на простые структуры и метаданные — теги, аннотации, связи.

9. Должна быть обеспечена ассоциативная навигация по ресурсам — формирование подсказок (ссылок) на схожие ресурсы.

10. Возможность полнотекстового поиска, а также поиска по ресурсам с учетом их структуры.

11. Возможность настройки интерфейсов просмотра и редактирования ресурсов.

12. Возможность работы в системе с различных устройств. Облачное хранение созданного информационного пространства пользователя.

13. Возможность совместной работы с ресурсами — обеспечение комментирования и обсуждения для пользователей системы, которым полное или частичное право предоставлено или делегировано владельцем ресурса.

Информационное пространство пользователя

Информационное пространство — это совокупностью всех сведений, с которыми человек работает сейчас или работал ранее. Любые данные, имеющие важность для человека и выделяемые им в отдельную сущность, рассматриваются как элементы информационного пространства — информационные ресурсы. Информационными ресурсами могут быть файлы, заметки, ранее посещенные веб-страницы, письма и пр. В рамках системы в качестве информационных ресурсов рассматриваются RDF-ресурсы. Таким образом, каждый пользователь в рамках системы ведет работу с собственным RDF-репозиторием данных, представляющим его информационное пространство. Структура информационного пространства описывается с помощью OWL-онтологии пользователя. По умолчанию предоставляется системная онтология, которая вводит ряд понятий, наиболее часто используемых пользователями, таких как Project, Person, Publication, File, E-mail и др. При необходимости пользователь может расширять стандартную онтологию, добавляя в нее новые классы и свойства или переопределяя уже имеющиеся. Наполнение репозитория происходит либо средствами автоматического импорта информации, либо вручную, через пользовательский интерфейс системы.

Категоризация

Деление файлов на различные иерархические категории является одним из основополагающих процессов, используемых пользователями при работе с информацией на персональных компьютерах. Тем не менее исследования показывают [16], что несмотря на понимание того, что категоризация в дальнейшем может существенно облегчить поиск, многие пользователи по различным причинам игнорируют эту возможность. В качестве объяснения такого поведения пользователи обычно ссылаются на сложности при принятии решения о том, в какую из категорий отнести файл; на вопросы, которые возникают при формировании подкатегорий, таких, чтобы их содержимое не пересекалось; на нехватку времени. По этой причине возможность автоматической или полуавто-

матической категоризации сведений, попадающих в систему, является крайне важной.

Необходимым элементом для проведения категоризации является выбор категорий, на которые будут делиться ресурсы. В ряде случаев возможно выбрать их полностью автоматически. Это относится к ресурсам, набор категорий для которых может быть получен из внешних источников. Например, научные статьи делятся на категории по тематикам работы, музыка и фильмы — по жанрам и направлениям.

Поскольку в общем случае автоматическое выделение категорий невозможно, формировать необходимые для последующего анализа классы можно по мере работы пользователя с системой. В таком случае категоризация может проводиться по следующим двум направлениям:

- выделение в общие категории ресурсов, находящихся в общих разделах иерархической структуры, организованной пользователем;
- выделение в общие категории на основании добавленных пользователем метаданных, таких как "теги".

За счет такой категоризации при добавлении нового ресурса в систему на основании внесенных ранее пользователем сведений ему будет предложено возможное расположение нового ресурса в иерархической структуре, а также метаданные, которые могут быть добавлены к этому ресурсу. В настоящее время существует довольно много различных методов категоризации текстовых данных. Наиболее популярные из них — наивный байесовский классификатор (*Naive Bayes Classifier*), метод опорных векторов (*Support Vector Machine*), метод "ближайшего соседа" (*k-nearest neighbour*). Описанные алгоритмы являются самообучающимися и в качестве входных данных для обучения получают набор категорий и распределенных по ним документов.

Интеллектуальный анализ данных

Одной из наиболее важных функций в системах управления личной информацией является возможность хранить метаданные о созданных в системе ресурсах. Хорошо известно, что обычно люди забывают или затрудняются заносить метаданные вручную, поэтому важно, чтобы система могла сформировать максимальное количество метаданных в автоматическом режиме. Как было отмечено ранее, часть метаданных формируется адаптерами к источникам данных на основании содержимого импортируемого ресурса. Кроме того, в ряде случаев можно получить дополнительную информацию из глобальной сети, для чего системой предоставляется ряд *адаптеров к внешним репозиторям*. Для имеющихся в системе ресурсов адаптеры осуществляют поиск аналогов в различных репозиториях глобальной сети (в частности, в репозиториях Linked Open Data). В случае успеха они переносят соответствующую информацию из найденных ресурсов в репозиторий пользователя. Каждый адаптер отвечает за поиск аналогов одного или нескольких классов OWL-онтологии пользователя.

В качестве примера рассмотрим адаптер для класса научных публикаций. Адаптер осуществляет поиск сведений о добавленных в систему статьях в онлайн-репозиториях публикаций (CiteCeerX, DBPL). Один из возможных способов реализации поиска рассмотрен в работе [17]. В случае успешного поиска в систему подгружается информация об авторах статьи и библиографическая информация, такая как цитирование, список литературы и пр. Для того чтобы сделать поиск более быстрым и эффективным, в рамках системы планируется поддерживать единый RDF-репозиторий публикаций, содержащий информацию о статьях, полученную из онлайн-репозиториев.

Другим аспектом анализа данных является поиск похожих или связанных ресурсов. Для каждого ресурса, находящегося в информационном пространстве пользователя, осуществляется поиск связанных с ним ресурсов как внутри информационного пространства, так и вне его — в глобальной сети. Алгоритмы поиска схожих ресурсов могут сильно отличаться в зависимости от класса искомого ресурса. Поэтому за поиск похожих ресурсов для разных классов отвечают различные компоненты.

Приведем пример алгоритма поиска новых научных публикаций на основании уже загруженных в систему. Для решения этой задачи используется определенный в рамках системы единый репозиторий публикаций. В репозитории каждой паре статей назначается коэффициент "схожести", вычисленный на основании методов анализа цитирования [18]. Для поиска публикаций, схожих с группой статей, в таком репозитории все хранящиеся в нем публикации разбивают на кластеры в соответствии с вычисленными значениями "схожести". Для этого репозиторий представляется в виде графа публикаций, в котором каждому ребру графа соответствует значение "схожести" его вершин. Кластеризация графа проводится на основании одного из распространенных алгоритмов [19]. После разбиения репозитория на кластеры по ним разбиваются статьи, загруженные пользователем, и в качестве схожих выдаются ближайшие статьи из того же кластера.

Реализация

В рамках данной статьи описан прототип системы, поддерживающий хранение и анализ научных публикаций. Прототип соответствует определенной выше архитектуре. В основе системы лежит RDF-база данных Sesame. Уровень адаптеров данных представляет адаптер файловой системы. На уровне сервисов реализован сервис анализа публикаций, отвечающий за пополнение сведений о публикациях данными из Академии Google (Google Scholar), а также осуществляющий в ней поиск новых публикаций, схожих с загруженными ранее.

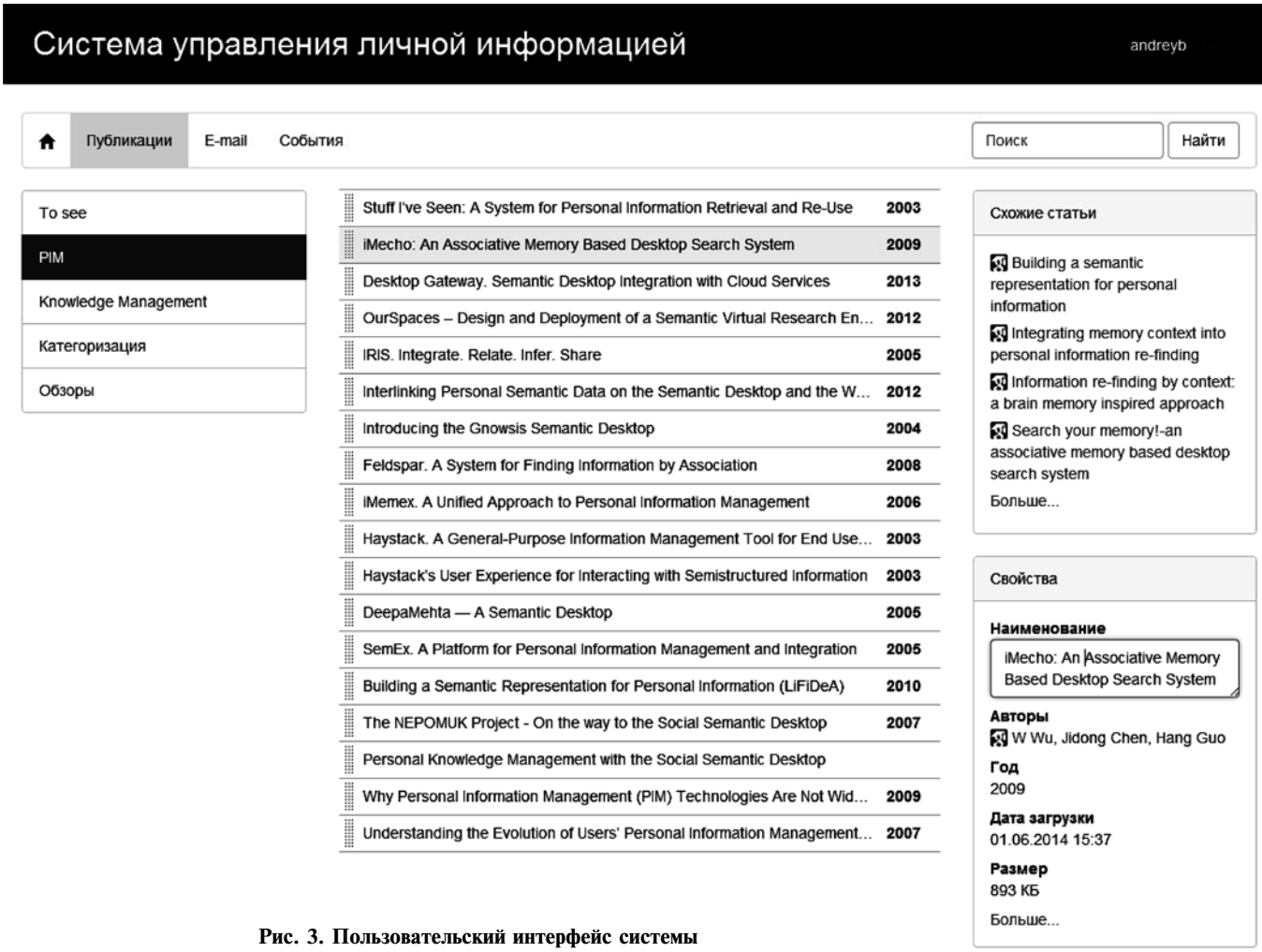


Рис. 3. Пользовательский интерфейс системы

Адаптер файловой системы представляет собой приложение, запущенное на компьютере пользователя, которое отвечает за передачу публикаций с компьютера пользователя на удаленный сервер. Приложение работает в фоновом режиме и при изменениях содержимого папок передает информацию об этом на сервер. На стороне сервера на основании текстового содержимого выгруженных файлов осуществляется поиск статей в Академии Google. В случае успешного поиска в систему заносятся метаданные, такие как название, год, авторы, а также ссылки на схожие статьи.

Работа пользователя с его информационным пространством осуществляется через веб-интерфейс. На рис. 3 представлен пользовательский интерфейс системы. Верхнее меню отвечает за навигацию по типам ресурсов, а также предоставляет возможность поиска ресурсов по системе. Рабочая область портала поделена на три части. В левом меню выводятся папки, синхронизованные с системой. В центральном блоке выводится список публикаций, находящихся в выбранной папке. Публикации, находящиеся в центральном блоке, могут быть отсортированы средствами интерфейса Drag&Drop. В правой части рабочей области выводится информация о статье, выбранной в центральном блоке, — схожие статьи и детальная информация. В панели схожих статей выводятся ссылки на статьи, схожие с выбранной публикацией. На панели детальной информации выводятся метаданные о выбранной публикации. Все поля, выводимые на панели детальной информации, при необходимости могут быть отредактированы пользователем.

Реализация прототипа выполнена на языке Java с использованием библиотек Spring, Sesame, Jena, jQuery, AngularJS, Twitter Bootstrap.

Апробация прототипа проводилась среди сотрудников и аспирантов ВЦ РАН. В ходе тестовой эксплуатации система показала хорошие результаты в части добавления метаданных к статьям, загруженным пользователями. Из около 300 загруженных в систему англоязычных статей примерно 85 % были найдены в Академии Google. В ходе тестирования со стороны ученых, которые принимали в нем участие, были высказаны некоторые замечания. К их числу относятся следующие: отсутствие возможности добавления к статьям заметок или других произвольных метадан-

ных; отсутствие возможности проставления связей между различными публикациями.

В дальнейшем планируется доработать прототип системы с учетом высказанных замечаний, а также добавить возможность выгрузки в систему других типов данных (e-mail, события, файлы произвольного формата). Важным направлением дальнейшей работы является поддержка взаимодействия между различными пользователями. Система должна обеспечивать возможность обсуждения и совместной работы с ресурсами.

Заключение

В статье были рассмотрены задачи управления личной информацией и знаниями, определены основные процессы, возникающие в ходе выполнения этих задач, сформулированы возможные направления автоматизации выявленных процессов. На основании анализа существующих работ были представлены основные требования, предъявляемые к системе, автоматизирующей работу пользователей с личной информацией. Была представлена базовая архитектура такой системы, в основе которой заложено использование семантических технологий, таких как RDF и OWL. В качестве реализации был представлен прототип системы, поддерживающий работу с научными публикациями. В рамках прототипа реализованы модули каждого из описанных уровней архитектуры — адаптер к файловой системе, сервис анализа публикаций, сервис поиска. За хранение ресурсов, загруженных пользователями, отвечает RDF — база данных Sesame.

В дальнейшем большее внимание планируется уделить созданию формальной модели системы, явно описывающей основные модули и процессы, выполняемые в рамках системы. Другим направлением работ является введение элементов логического вывода в рамках анализа информационного пространства. Поскольку информация хранится в формате RDF и описывается с помощью языка OWL, принципиальных ограничений в этом вопросе нет.

Список литературы

1. **Bush V.** As We May Think // The Atlantic Monthly. 1945. Vol. 176. P. 101—108.

2. **Alavi M., Leidner D. E.** Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues // MIS quarterly. 2001. Vol. 25, N. 1. P. 107—136.

3. **Alavi M., Leidner D. E.** Knowledge management systems: issues, challenges, and benefits // Communications of the AIS. 1999. Vol. 1, N. 7. P. 1—27.

4. **Nonaka I.** A dynamic theory of organizational knowledge creation // Organization science. 1994. Vol. 5, N. 1. P. 14—37.

5. **Sauermann L., Grimnes G.A., Kiesel M.** et al. Semantic desktop 2.0: The gnowsis experience // The Semantic Web — ISWC 2006. Berlin: Springer, 2006. P. 887—900.

6. **Dong X. L., Halevy A.** A platform for personal information management and integration // In Proceedings of CIDR 2005. Asilomar, CA, USA. January 4—7, 2005. URL:http://www.cidrdb.org/cidr2005/papers/P10.pdf

7. **Cheyer A., Park J., Giuli R.** IRIS: Integrate, Relate. Infer. Share // In Proceedings of the Semantic Desktop Workshop at ISWC. Galway, Ireland. November 6, 2005. URL: http://ceur-ws.org/Vol-175/17_park_iris_final.pdf

8. **Karger D. R., Bakshi K., Huynh D.** et al. Haystack: A General-Purpose Information Management Tool for End Users Based on Semistructured Data // In Proceedings of CIDR 2005, Asilomar, CA, USA. January 4—7, 2005. URL: http://www.cidrdb.org/cidr2005/papers/P02.pdf

9. **Gemmell J., Bell G., Lueder R.** et al. MyLifeBits: fulfilling the Memex vision // Proceedings of the tenth ACM international conference on Multimedia. Juan les Pins, France. December 1—6, 2002. New York, USA: Association for Computing Machinery, 2002. P. 235—238.

10. **Richter J., Völkel M., Haller H.** DeepaMehta — A Semantic Desktop // In Proceedings of the Semantic Desktop Workshop at ISWC Galway, Ireland. November 6, 2005. URL: http://ceur-ws.org/Vol-175/30_dm_poster.pdf

11. **Chirita P. A., Costache S., Nejdl W.** et al. Beagle++: Semantically enhanced searching and ranking on the desktop // The Semantic Web: Research and Applications. Berlin: Springer, 2006. P. 348—362.

12. **Chen J., Guo H., Wu W., Wang W.** iMecho: an associative memory based desktop search system // CIKM '09: Proceedings of the 18th ACM Conference on Information and Knowledge Management. Hong Kong, China. November 2—6, 2009. New York, USA: ACM, 2009. P. 731—740.

13. **Chau D. H., Myers B., Faulring A.** What to Do when Search Fails: Finding Information by Association // Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. Florence, Italy. April 5—10, 2008. New York, NY, USA:ACM, 2008. P 999-1008.

14. **Kareski A., Jovanovik M., Trajanov D.** Desktop Gateway: Semantic Desktop Integration with Cloud Services // BCI '13: Proceedings of the 6th Balkan Conference in Informatics. Thessaloniki, Greece. September 19—21, 2013. New York, USA: ACM, 2013. P 162—168.

15. **Hitzler P., Krötzsch M., Parsia B.,et al.** OWL 2 web ontology language primer. W3C recommendation. 2012. URL: http://www.w3.org/TR/owl2-primer/

16. **Voit K., Andrews K., Slany W.** Why personal information management (pim) technologies are not widespread // ASIS&T 2009 Workshop on Personal Information Management. Vancouver, BC, Canada November 7—8, 2009. URL: http://pimworkshop.org/2009/papers/voit-pim2009.pdf

17. **Aumüller D., Rahm E.** PDFMeat: managing publications on the semantic desktop // CIKM'11: Proceedings of the 20th ACM international conference on Information and knowledge management. Glasgow, United Kingdom. October 24—28, 2011. New York, USA: ACM, 2011. P. 2565—2568.

18. **Boyack K.W., Klavans R.** Co-citation analysis, bibliographic coupling, and direct citation: Which citation approach represents the research front most accurately? // Journal of the American Society for Information Science and Technology. 2010. Vol. 61, N. 12. P. 2389—2404.

19. **Schaeffer S. E.** Graph clustering // Computer Science Review. 2007. Vol. 1. N. 1. P. 27—64.

A.A. Bezdushny, Postgraduate Student, Moscow Institute of Physics and Technology,
e-mail: andrey.bezdushny@gmail.com

Conceptual Provisions and Architecture of Semantic Personal Information Management System

This paper considers the problem of the personal information management by using semantic technologies, proposes the architecture of semantic personal information management systems and presents the prototype of the system implemented in accordance with this architecture. Proposed method develops the idea of the Semantic Desktop — an approach to the personal information space organization in accordance with Semantic Web and Linked Open Data.

Within the system, information space is considered as RDF repository, based on OWL ontology. By default, system OWL ontology, containing the most frequently used concepts, is available to user. If necessary, user can extend it by adding new classes and properties or overriding existing ones.

Following are the main objectives of the system. Maintenance of user information space and structuring of resources within it. Harvesting of the information stored in external sources such as file system, mail server etc. Support for the search through the information space. Automated analysis of the information space, in particular harvesting of additional metadata from the web. Automated resources categorization. Support for collaboration.

Keywords: *personal information management, personal knowledge management, Semantic Desktop, Semantic Web, RDF, Linked Open Data, data analysis, data categorization*

References

1. **Bush V.** As We May Think. *The Atlantic Monthly*. 1945. Vol. 176. P. 101—108.

2. **Alavi M., Leidner D. E.** Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues. *MIS quarterly*. 2001. Vol. 25. N. 1. P. 107—136.

3. **Alavi M., Leidner D. E.** Knowledge management systems: issues, challenges, and benefits. *Communications of the AIS*. 1999. Vol. 1, N. 7. P. 1—27.

4. **Nonaka I.** A dynamic theory of organizational knowledge creation. *Organization science*. 1994. Vol. 5, N. 1. P. 14—37.

5. **Sauermann L., Grimnes G.A., Kiesel M.** et al. Semantic desktop 2.0: The gnowsis experience. *The Semantic Web — ISWC 2006*. Berlin: Springer, 2006. P. 887—900.

6. **Dong X. L., Halevy A.** A platform for personal information management and integration. *In Proceedings of CIDR 2005*. Asilomar, CA, USA. January 4—7, 2005. URL:http://www.cidrdb.org/cidr2005/papers/P10.pdf

7. **Cheyer A., Park J., Giuli R.** IRIS: Integrate, Relate. Infer. Share. *In Proceedings of the Semantic Desktop Workshop at ISWC*. Galway, Ireland. November 6, 2005. URL: http://ceur-ws.org/Vol-175/17_park_iris_final.pdf

8. **Karger D. R., Bakshi K., Huynh D.** et al. Haystack: A General-Purpose Information Management Tool for End Users Based on Semistructured Data. *In Proceedings of CIDR 2005*, Asilomar, CA, USA. January 4—7, 2005. URL: http://www.cidrdb.org/cidr2005/papers/P02.pdf

9. **Gemmell J., Bell G., Lueder R.** et al. MyLifeBits: fulfilling the Memex vision. *Proceedings of the tenth ACM international conference on Multimedia*. Juan les Pins, France. December 1—6, 2002. New York, USA: Association for Computing Machinery, 2002. P. 235—238.

10. **Richter J., Völkel M., Haller H.** DeepaMehta — A Semantic Desktop. *In Proceedings of the Semantic Desktop Workshop at ISWC Galway*. Ireland. November 6, 2005. URL: http://ceur-ws.org/Vol-175/30_dm_poster.pdf

11. **Chirita P. A., Costache S., Nejdl W. et al.** Beagle++: Semantically enhanced searching and ranking on the desktop. *The Semantic Web: Research and Applications*. Berlin: Springer, 2006. P. 348—362.

12. **Chen J., Guo H., Wu W., Wang W.** iMecho: an associative memory based desktop search system. *CIKM '09: Proceedings of the 18th ACM Conference on Information and Knowledge Management*. Hong Kong, China. November 2—6, 2009. New York, USA: ACM, 2009. P. 731—740.

13. **Chau D. H., Myers B., Faulring A.** What to Do when Search Fails: Finding Information by Association. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Florence, Italy. April 5—10, 2008. New York, NY, USA:ACM, 2008. P 999-1008.

14. **Kareski A., Jovanovik M., Trajanov D.** Desktop Gateway: Semantic Desktop Integration with Cloud Services. *BCI '13: Proceedings of the 6th Balkan Conference in Informatics*. Thessaloniki, Greece. September 19—21, 2013. New York, USA: ACM, 2013. P 162—168.

15. **Hitzler P., Krötzsch M., Parsia B.,et al.** OWL 2 web ontology language primer. W3C recommendation. 2012. URL: http://www.w3.org/TR/owl2-primer/

16. **Voit K., Andrews K., Slany W.** Why personal information management (pim) technologies are not widespread. *ASIS&T 2009 Workshop on Personal Information Management*. Vancouver, BC, Canada. November 7—8, 2009. URL: http://pimworkshop.org/2009/papers/voit-pim2009.pdf

17. **Aumüller D., Rahm E.** PDFMeat: managing publications on the semantic desktop. *CIKM'11: Proceedings of the 20th ACM international conference on Information and knowledge management*. Glasgow, United Kingdom — October 24—28, 2011. New York, USA: ACM, 2011. P. 2565—2568.

18. **Boyack K.W., Klavans R.** Co-citation analysis, bibliographic coupling, and direct citation: Which citation approach represents the research front most accurately? *Journal of the American Society for Information Science and Technology*. 2010. Vol. 61, N. 12. P. 2389—2404.

19. **Schaeffer S. E.** Graph clustering. *Computer Science Review*. 2007. Vol. 1. N. 1. P. 27—64.

УДК 004.925.8

В. Д. Аввакумов, канд. техн. наук, доц., e-mail: awwam@mail.ru,
Новоуральский технологический институт НИЯУ "МИФИ"

Аппроксимация фигур сложной геометрической формы с использованием метода стыковки

Представлены алгоритмы нахождения простых фигур минимальной площади и выпуклой оболочки, аппроксимирующих плоские фигуры сложной геометрической формы. Алгоритмы строятся на основе исходных данных, которые получаются разработанным автором методом стыковки: определяется положение фигуры при ее поступательном движении в заданном направлении к неподвижной фигуре до контакта с ней. При поиске аппроксимирующих фигур могут учитываться допуски на размеры заданной фигуры. Приведен пример нахождения различных аппроксимирующих фигур для заданной фигуры. Рассматриваемые алгоритмы позволяют находить аппроксимирующую фигуру с наибольшим коэффициентом заполнения при решении задач оптимизации размещения фигур сложной формы.

***Ключевые слова**: аппроксимация, фигуры сложной геометрической формы, простые фигуры, выпуклая оболочка, контур, алгоритмы, непрерывно-дискретное представление контура, нестинг, опорная прямая, допуск на размер*

Введение

В практике конструкторско-технологического проектирования существуют задачи, решение которых направлено на оптимальное размещение фигур сложной геометрической формы в заданных областях плоскости — так называемые проблемы нестинга [1]. Эти задачи встречаются в самых разнообразных областях науки и техники: в проектировании машин и механизмов, радиоэлектронных устройств, интегральных схем, технологических процессов, в физике и химии.

Проблемы нестинга относятся к классу NP-трудных задач комбинаторной оптимизации, т. е. для их точного решения необходим полный перебор возможных вариантов, что требует значительных вычислительных ресурсов. Самой трудоемкой в решении таких задач является процедура проверки условия взаимного непересечения фигур при их перемещении относительно друг друга. Для упрощения этой задачи исходную фигуру сложной геометрической формы аппроксимируют простой фигурой. Такими наиболее часто применяемыми простыми фигурами являются прямоугольник или многоугольник. Однако существуют фигуры, аппроксимация которых фигурой другой геометрии дает лучшее решение (в смысле отношения площади аппроксимирующей фигуры к исходной фигуре). Это могут быть такие фигуры, как параллелограмм, трапеция, окружность и другие простые фигуры.

Известны работы, посвященные нахождению некоторых простых фигур, аппроксимирующих заданную фигуру. Прямоугольная аппроксимация рассматривается в работах [1—3]. В работах [4, 5] плоский контур аппроксимируется многоугольником. Работа [6] содержит методы отыскания параллелограмма минимальной площади, описывающего заданный объект. В работе [7] изложен алгоритм отыскания окружности минимальной площади, описанной около контура или группы контуров.

При решении задач машинной графики, распознавания образов и изображений, автоматизации проектирования, технологических процессов (в частности, размещения, раскроя-упаковки) требуется для плоских геометрических фигур найти выпуклую оболочку. Построение выпуклых оболочек — очень важная процедура в ряде приложений не только сама по себе, она может являться частью других алгоритмов, использующих эту процедуру в качестве составляющей. Известно большое число работ, посвященных построению выпуклых оболочек конечного множества точек на плоскости и простых многоугольников. Для фигур произвольной геометрической формы задача нахождения выпуклой оболочки является более сложной по сравнению с задачами аппроксимации простыми фигурами. Автору известна только одна работа, посвященная построению выпуклой оболочки для реальных контуров деталей [7].

В настоящее время не существует единого подхода к решению задач аппроксимации фигур сложной геометрической формы. Для каждого алгоритма в известных работах используется, во-первых, своя входная геометрическая информация о заданной фигуре и выходная информация об аппроксимирующих ее фигурах, во-вторых, свой методологический аппарат.

В данной работе предлагается разработанная автором технология решения задач аппроксимации фигур сложной формы с использованием единой входной геометрической информации и получением единой выходной геометрической информации на основе об-

щего методического подхода. При этом рассматриваются следующие задачи аппроксимации:

1) построение простых фигур минимальной площади;

2) построение выпуклой оболочки.

Понятия, используемые в работе

Рассмотрим понятия, используемые в данной работе. Одни из них являются общеизвестными (при их формулировке делается ссылка на соответствующий источник), другие введены автором.

Прямую, которая содержит по крайней мере одну точку контура фигуры, но не содержит ее внутренних точек, называют *опорной прямой* (ОП) фигуры [8]. В предлагаемой работе ОП — плоская прямоугольная фигура, имеющая достаточную длину (превосходящую бóльший габаритный размер фигуры) и малую толщину (толщина ОП роли не играет). *Опорной функцией* $h(\theta)$ фигуры называют зависимость расстояния от полюса до опорной прямой, проведенной к фигуре, от угла θ, определяющего направление движения опорной прямой к фигуре [9]. *Полюсом* будем считать центр тяжести фигуры.

В геометрии *простой* считается фигура, если ее можно разбить на конечное число треугольников. В работе рассматриваются такие простые фигуры, контуры которых представляют собой треугольник, прямоугольник, параллелограмм, трапецию. В общем случае — это выпуклые многоугольники (*n-угольники*). К простым фигурам также отнесем *окружность*.

Фигуру минимальной площади, в которой целиком находится заданная фигура, называют *выпуклой оболочкой* [8].

Простые фигуры минимальной площади и выпуклую оболочку будем называть *аппроксимирующими фигурами* (АФ). Каждая АФ представляется *контуром*, состоящим в общем случае из замкнутой последовательности соединенных между собой геометрических элементов — отрезков прямых линий и дуг окружностей. В свою очередь, контур определяется *характерными* точками — последовательностью *узловых* точек и точек — *центров дуг*. Узловые точки могут быть *угловыми* и точками *сопряжения* (рис. 1).

Выходными данными алгоритмов нахождения АФ являются массивы *дискретно-непрерывного представления* (ДНП) контура: $X = (x_1, x_2, ..., x_n)$, $Y = (y_1, y_2, ..., y_n)$, где x_i, y_i — координаты характерных точек контура ($i = \overline{1, n}$); n — число характерных точек в контуре. Нумерация характерных точек контура АФ осуществляется при движении по контуру против часовой стрелки числами от 1 до n в порядке их расположения в контуре.

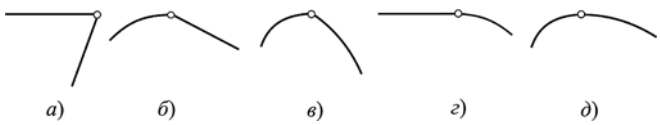


Рис. 1. Варианты соединения пары элементов контура АФ:
а, *б*, *в* — угловые точки; *г*, *д* — точки сопряжения

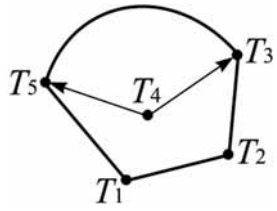


Рис. 2. Контур АФ, содержащий дугу

Если при этом в контуре имеются дуги, то после начальной точки каждой дуги идет номер точки — центра этой дуги (рис. 2).

При использовании массивов ДНП контура определить точку — центр дуги можно по результатам вычисления расстояния между парами точек T_{i-1}, T_i и T_i, T_{i+1} и по ее расположению: если эти расстояния равны и точка T_i лежит слева при движении по контуру против часовой стрелки, то точка T_i является искомым центром дуги. На рис. 2 отрезки T_1T_2 и T_2T_3 равны, но точка T_2 лежит справа при движении по контуру, следовательно, она не является центром дуги. Центром дуги является точка T_4, так как $T_3T_4 = T_4T_5$ и она находится слева при движении по контуру.

Методологическая основа решения задач аппроксимации и формирование исходных данных

Методологической основой предлагаемых автором алгоритмов решения задач аппроксимации фигур сложной геометрии является метод *стыковки*, разработанный автором [10]. Суть данного метода заключается в определении положения плоской фигуры при ее поступательном движении в заданном направлении к неподвижной фигуре до контакта с ней, если такой контакт возможен.

Особенностью метода стыковки является то, что для решения этой задачи не требуется аналитического описания и вывода условий взаимного непересечения объектов в виде неравенств. Описание фигур осуществляется с помощью специально разработанного автором параметрического языка ПЛОГАС [11], состоящего из операторов плоских геометрических построений, которые реализуют функциональную связь координат характерных точек контура с размерами-параметрами P_i: $x_j = x_j(P_1, P_2, ..., P_n)$ и $y_j = y_j(P_1, P_2, ..., P_n)$.

Алгоритмы, реализующие рассматриваемые движения, построены таким образом, что подвижная фигура может только касаться неподвижной, но не пересекать ее. Таким образом, автоматически реализуется условие непересечения фигур. При этом находятся точки контакта фигур (*контактные точки*). Следует заметить, что контактных точек может быть одна, несколько, бесконечное множество, ни одной, если в заданном направлении перемещения подвижная фигура "проскакивает" неподвижную. Если точек контакта несколько, то, как принято в методе стыковки, всегда вы-

бирается крайняя *левая*, если смотреть на неподвижную фигуру в направлении перемещения подвижной.

В методе стыковки используется дискретно-непрерывный подход: область возможного контакта контуров фигур разделяется на множество пар линий их контуров, состоящих из отрезков прямых и дуг окружностей (*отрезок—отрезок*, *отрезок—дуга*, *дуга—дуга*). Между этими парами с помощью геометрических построений (с использованием соответствующих аналитических выражений) находятся локальные минимумы расстояний, среди которых выбирается глобальный минимум. Такой подход позволяет получить быстрый, точный и устойчивый алгоритм поиска глобального минимума [10].

Метод стыковки может применяться как для фигуры, состоящей из одного контура, так и из *группы* замкнутых непересекающихся контуров, каждый из которых состоит из отрезков прямых и/или дуг окружностей. Группа контуров представляет собой совокупность взаимосвязанных контуров, имеющих или не имеющих общие точки. Взаимосвязь контуров определяется их описанием (на языке ПЛОГАС) в единой системе координат. Примером группы контуров может быть сечение какой-либо сборочной единицы.

Исходными данными для решения сформулированных выше задач аппроксимации являются: для первой — опорная функция, для второй — множество контактных точек. Рассмотрим общую схему формирования опорной функции и множество контактных точек, используя метод стыковки [10]. Для пояснения операций стыковки применим следующую механическую аналогию. Нормаль к ОП представим в виде невесомого стержня, шарнирно закрепленного в полюсе фигуры (рис. 3).

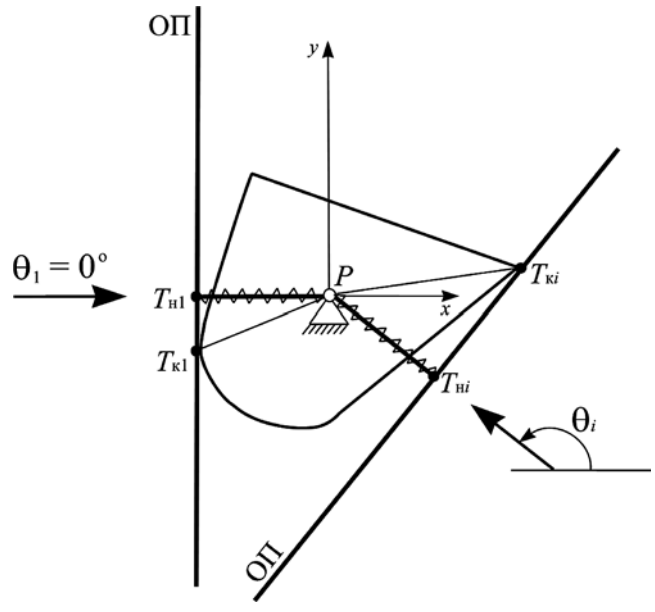


Рис. 3. Система вращающихся ОП и стержня

Через отверстие в середине опорной ОП она надевается на стержень и соединяется с полюсом пружиной, находящейся на стержне. При вращении системы ОП—стержень опорная прямая скользит вдоль стержня, оставаясь всегда прижатой к фигуре пружиной. Эта система вращается против часовой стрелки вокруг полюса с заданным шагом $\Delta\theta$. Следует отметить, что размер этого шага определяется той точностью, с которой требуется вычислять характеристики фигуры. Эта точность может диктоваться, например, размерами и/или допусками на размеры фигуры, если поиск АФ выполняется с учетом допусков.

После выполнения процедуры стыковки на каждом шаге вращения ОП вычисляются следующие параметры (рис. 3):

- координаты $x_н(\theta_i)$, $y_н(\theta_i)$ точки $T_{нi}$ конца нормали к ОП;
- расстояние $PT_{нi}$ от полюса P до ОП (значение опорной функции $h(\theta_i)$);
- координаты $x_к(\theta_i)$, $y_к(\theta_i)$ контактной точки $T_{кi}$ ОП с фигурой; $i = 1, 2, ..., m$; $m = 2\pi/\Delta\theta$.

Такая операция повторяется до совершения ОП полного оборота вокруг фигуры. В результате получаем табличную дискретную опорную функцию $h(\theta_i)$ и множество контактных точек $T_к = \{T_i\} = \{x_к(\theta_i), y_к(\theta_i)\}$.

Построение простых фигур минимальной площади

Имея опорную функцию $h(\theta)$, можно получить указанные выше простые фигуры минимальной площади (найти массивы ДНП контура). Для этого отдельно рассмотрим выпуклый n-угольник, при различных n которого находятся конкретные простые фигуры, и окружность, алгоритмы построения которых различны.

Выпуклый n-угольник. Найдем, прежде всего, выражение для вычисления площади охватывающего фигуру n-угольника (рис. 4), используя значения опорной функции (являющиеся длинами нормалей к его сторонам) и заданные значения $n - 1$ углов между нормалями α_j ($\alpha_n = 2\pi - \sum_{i=1}^{n-1} \alpha_i$).

Отметим, что число этих углов и их значения определяются видом многоугольника. Например, для получения аппроксимирующего равнобедренного прямоугольного треугольника задаются $\alpha_1 = \pi/2$ и $\alpha_2 = \pi/4$.

Для нахождения площади выпуклого n-угольника необходимо проводить выборку n значений нормалей из множества значений опорной функции $h(\theta_i)$. Введем следующие обозначения:

i — номер выборки ($i = 1, 2, ..., m_1$; $m_1 = m/n = 2\pi/(n\Delta\theta)$);

j — порядковый номер значения нормали в i-й выборке ($j = 1, 2, ..., n$);

$l(i, j)$ — порядковый номер значения опорной функции, определяемый значениями i и j.

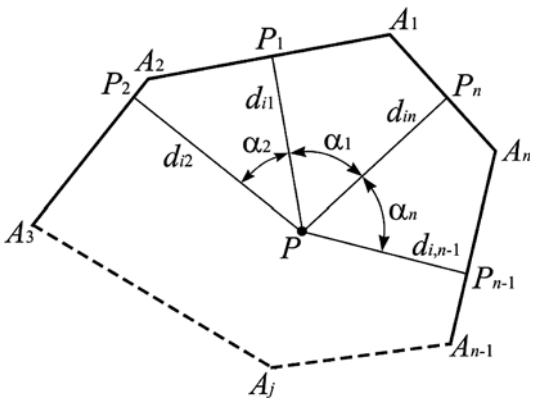


Рис. 4. Выпуклый *n*-угольник

Соотношение

$$l(i, j) = i + \frac{1}{\Delta\theta}\sum_{r=1}^{j-1}\alpha_r \qquad (1)$$

позволяет для каждой выборки i по углам α_r определять соответствующую совокупность нормалей $h_{l(i,j)} = h(\theta_{l(i,j)})$ (в дальнейшем для краткости будем обозначать $h(\theta_{l(i,j)}) = d_{ij}$). Например, для равнобедренного прямоугольного треугольника при $\Delta\theta = 1$ и $i = 1$ номера значений опорной функции будут следующими: $l(1,1) = 1$, $l(1,2) = 91$, $l(1,3) = 136$. Выборка таким образом будет состоять из элементов $d_{1,1} = h(\theta_1)$, $d_{1,2} = h(\theta_{91})$, $d_{1,3} = h(\theta_{136})$.

Рассмотрим выпуклый n-угольник, построенный по выборке нормалей с номером i (рис. 4). Его площадь равна сумме площадей n четырехугольников $PP_nA_1P_1$, $PP_1A_2P_2$, ..., $PP_{n-1}A_nP_n$. Если в каждом из этих четырехугольников соединить вершину (точку A_j) с полюсом (точкой P), то его площадь сведется к сумме площадей прямоугольных треугольников, выражения для которых легко выводятся через d_{ij} и α_j:

$$F_i = \frac{1}{2}\sum_{r=1}^{n}\left(d_{ir}^2\left(\frac{1 - q_{ir}\cos\alpha_r}{q_{ir}\sin\alpha_r}\right) + d_{jr}^2\left(\frac{q_{ir} - \cos\alpha_r}{\sin\alpha_r}\right)\right), \qquad (2)$$

где $q_{ir} = \dfrac{d_{ir}}{d_{jr}}$; d_{ir}, d_{jr} — значения нормалей; α_r — углы между нормалями; $j = i\left(1 - \left\lfloor\dfrac{i}{n}\right\rfloor\right) + 1$ (скобки $\lfloor\ \rfloor$ означают взятие целой части числа); $i, j = 1, 2, ..., n$.

Искомые выпуклые n-угольники можно разбить на две группы:

1) число сторон и углы α_i между нормалями заданы, т. е. вид фигуры предопределен;

2) число сторон не задано.

К первой группе относятся такие простые фигуры, как правильный n-угольник, n-угольник с заданными углами между нормалями (это могут быть, например, такие фигуры, как прямоугольник, прямоугольный треугольник с равными катетами и др.). Вторая группа содержит недоопределенные фигуры, для которых задан либо только вид, например, параллелограмм, либо — фиксированное число сторон многоугольника.

В общем случае для нахождения n-угольника используется одна процедура (назовем ее *первичной*) или две процедуры (первичная и *вторичная*). С помощью первичной процедуры находится n-угольник, относящийся к первой группе.

Вторичная процедура применяется для нахождения n-угольников минимальной площади второй группы и выполняется после первичной. Таким образом, для этих фигур с помощью первичной процедуры находится первое (грубое) приближение n-угольника. Применение вторичной процедуры позволяет получить второе (точное) приближение.

Рассмотрим алгоритмы первичной и вторичной процедур.

Первичная процедура. Алгоритм первичной процедуры следующий.

Шаг 1. Задаются: число сторон АФ; $i = 1$ — номер выборки значений нормалей из таблицы опорной функции; углы между нормалями к сторонам. В случае, когда не задан конкретный вид АФ (задано только число сторон n-угольника), принимают $\alpha_j = 2\pi/n$, $j = 1, 2, ..., n - 1$. Если требуется найти фигуру, имеющую конкретный вид, то значения α_j задаются в соответствии с этим видом. Например, если требуется найти прямоугольник минимальной площади, то $\alpha_1 = \alpha_2 = \alpha_3 = \pi/2$; если определяется прямоугольный треугольник, то $\alpha_1 = \pi/2$, $\alpha_2 = 3\pi/4$.

Шаг 2. По формуле (1) выполняется i-я выборка n значений нормалей и по формуле (2) вычисляется площадь F_i n-угольника.

Шаг 3. Полагается $i = i + 1$ и если $i \leqslant m_1$ ($m_1 = 2\pi/n\Delta\theta$), то переход к шагу 2.

Шаг 4. Среди полученных m_1 значений площадей n-угольника выбирается $F_{\min}$, т. е. становятся известными значения n нормалей, дающих n-угольник минимальной площади и его положение в системе координат, в которой описана фигура. Иначе говоря, определяется такой номер i^* выборки нормалей (множество $\{d_{i^*,1}, d_{i^*,2}, d_{i^*,3}, ..., d_{i^*,n}\}$), который соответствует $F_{\min}$.

По существу, приведенный алгоритм реализует вращение n-угольника с фиксированными углами между нормалями, опущенными из полюса на его стороны, вокруг неподвижной фигуры (вращение n-угольника проводится с шагом $\Delta\theta$ и таким образом, что все его стороны при каждом повороте касаются фигуры).

Если искомый n-угольник относится к первой группе, то задача считается решенной, т. е. найдены массивы ДНП контура аппроксимирующей фигуры.

Вторичная процедура. С помощью вторичной процедуры поочередно определяется положение каждой из сторон найденного первичной процедурой n-угольника, при котором его площадь становится минимальной: такое положение находится путем вращения выбранной стороны вокруг исходной фигуры (аналогично вращению системы ОП—стержень, см. рис. 3).

Рассмотрим алгоритм определения положения стороны, на которую опущена найденная первичной процедурой нормаль, соответствующая углу θ_j (ее длина $d_{i^*, j} = h(\theta_j)$).

Шаг 1. Отступлением на шаг назад и на шаг вперед от значения нормали $d_{i^*, j}$ выбирается то направление движения по функции $h(\theta)$, при котором площадь n-угольника будет уменьшаться.

Шаг 2. Значение θ_j меняется в соответствии с выбранным направлением ($\theta_{j+1} = \theta_j + \Delta\theta$ или $\theta_{j-1} = \theta_j - \Delta\theta$) и осуществляется выборка значения нормали: $d_{i^*, j+1} = h(\theta_{j+1})$ или $d_{i^*, j-1} = h(\theta_{j-1})$.

Шаг 3. По формуле (2) вычисляется площадь F_{i^*} с новым значением нормали и выполняется проверка: $F_{i^*} < F_{\min}$. Если да, то $F_{\min} = F_{i^*}$ и переход на шаг 2, иначе — переход к следующему шагу.

Шаг 4. За искомое принимается положение стороны n-угольника, определяемое нормалью при предыдущем значении угла θ_j.

Данный алгоритм выполняется для тех сторон найденного первичной процедурой n-угольника, при вращении которых можно получить более точное решение. Если таким n-угольником будет:

- произвольный треугольник, то поочередно вращаются все стороны найденного треугольника;
- прямоугольный треугольник, то вращается сторона, противолежащая прямому углу;
- параллелограмм, то у прямоугольника минимальной площади поочередно вращается первая пара параллельных сторон (вторая пара остается неподвижной), затем вращается вторая пара при неподвижной первой паре и среди двух получившихся фигур выбирается имеющая минимальную площадь;
- трапеция, то у прямоугольника минимальной площади вращается одна сторона, затем вращается вторая, параллельная ей сторона; процедура повторяется для второй пары параллельных сторон и из четырех площадей выбирается минимальная;
- произвольный n-угольник ($n > 4$), то поочередно вращаются все его стороны.

Таким образом, с помощью первичной процедуры или первичной и вторичной процедур находится совокупность нормалей и углов между ними, определяющих выпуклый n-угольник минимальной площади. Одновременно со значениями найденных нормалей становятся известными и координаты их концов $x_н(\theta_i)$, $y_н(\theta_i)$, вычисленные операцией стыковки. Эти координаты совместно с координатами полюса позволяют найти координаты вершин n-угольника — точек пересечения линий, совпадающих с опорными прямыми (на рис. 4 это линии, проходящие через точки A_1 и A_2, A_2 и A_3, ..., A_n и A_1). Тем самым определяется положение найденного n-угольника относительно заданной фигуры, т. е. находятся массивы ДНП контура.

Окружность. Для построения окружности требуется найти ее диаметр, который также как и рассмотренное выше построение n-угольника, определяется по значениям опорной функции $h(\theta)$:

$$D(\theta^*) = \max_{i \in 1, m} (h(\theta_i) + h(\theta_i + \pi)).$$

По найденному углу θ^* находятся координаты центра искомой окружности. Для этого по значениям θ^* и $\theta^* + \pi$ выбираются координаты контактных точек ОП с фигурой ($x_к(\theta^*)$, $y_к(\theta^*)$) и ($x_к(\theta^* + \pi)$, $y_к(\theta^* + \pi)$) — концов диаметра окружности. Затем определяется угол наклона диаметра и от начальной его точки под найденным углом откладывается радиус окружности; координаты этой точки и будут определять искомый центр окружности минимальной площади, аппроксимирующей исходную фигуру.

Построение выпуклой оболочки

Перейдем к рассмотрению второй задачи аппроксимации: построению выпуклой оболочки. Отметим, что множество контактных точек ОП с исходной фигурой $T_к = \{x_к(\theta_i), y_к(\theta_i)\}$ ($i = 1, 2, ..., m$), полученное методом стыковки (см. рис. 3), есть дискретное представление выпуклой оболочки, так как все контактные точки в соответствии с алгоритмом их получения принадлежат ее контуру. Однако это множество содержит значительный объем избыточной информации, связанный с пошаговым нахождением контактных точек, вследствие чего возникает необходимость в представлении его в более компактном виде. Основой такого представления является то, что контур выпуклой оболочки, как и контуры исходной фигуры, в общем случае состоит из таких элементов, как соединенные между собой отрезки прямых и дуги окружностей, т. е. является дискретно-непрерывным. Таким образом, для более компактного представления полученного дискретного контура выпуклой оболочки необходимо заменить его на дискретно-непрерывное. Анализ показал, что оптимальный вариант такого представления (содержащего минимальный объем информации об элементах контура) будет реализован, если информацию о контуре выпуклой оболочки записать в массивы ДНП контура этой оболочки.

Таким образом, задача построения выпуклой оболочки сводится к получению массивов ДНП ее контура на основе множества контактных точек $T_к$, иначе говоря, к нахождению координат узловых точек и, если контур исходной фигуры имеет дуги, координат центров дуг. Рассмотрим алгоритм реализации данной задачи.

Шаг 1. Подготовка исходных данных:

- записываются в файл координаты полюса, размер шага $\Delta\theta$ по углу вращения ОП, координаты контактных точек множества $T_к$;
- $i = 0$; $ip = 0$.

Шаг 2. Если во множестве $T_к$ есть подмножества угловых точек (имеющих совпадающие координаты), то из этих подмножеств удаляются все точки, за исключением последней. В результате получается новое множество контактных точек $T'_к = \{x_к(\theta_i), y_к(\theta_i)\}$; $i = 1, 2, ..., l$, где l — число оставшихся во множестве $T'_к$ контактных точек.

Шаг 3. Полагается $i = i + 1$ и делается проверка $i > l$. Если $i > l$, то переход к шагу 6. Иначе выбирается пара соседних точек T_i, T_{i+1} из множества $T'_к$ (если $i = l$, то выбирается пара точек T_l, T_1).

Шаг 4. Проверяется, являются ли выбранные точки началом и концом отрезка прямой. Для этого по формуле косинусов вычисляется угол α между прямыми, соединяющими полюс с этими точками (рис. 5, *а*). Если $\alpha > \Delta\theta$, то проводятся следующие действия:

- в массивах ДНП контура запоминаются координаты начальной точки T_i отрезка;
- полагается $ip = 1$ — признак того, что выбранные точки соединены отрезком;
- переход к шагу 3.

Если $\alpha = \Delta\theta$ (выбранные точки принадлежат дуге), то переход к следующему шагу.

Шаг 5. Проверка, является ли точка T_i начальной точкой дуги. Это может быть в двух случаях:

1) когда предыдущая пара точек является началом и концом отрезка, т. е. $ip = 1$ (рис. 5, *б*);

2) центр дуги, вычисленный по предыдущей паре точек дуги, не совпадает с центром дуги (с заданной точностью), вычисленным по текущей паре точек дуги (рис. 5, *в*); иначе говоря, точки T_{i-1}, T_i, T_{i+1} не принадлежат одной дуге. Центр дуги находится как точка пересечения линий направления движения ОП к фигуре в паре точек T_i, T_{i+1} или T_{i-1}, T_i, (рис. 5, *б*, *в*).

Если имеет место тот или иной случай, выполняются следующие действия:

- в массивах ДНП контура запоминаются координаты начальной точки T_i дуги и координаты центра дуги, которой принадлежат точки T_i, T_{i+1} (рис. 5, *б*, *в*);
- переход к шагу 3.

Иначе (точки T_{i-1}, T_i, T_{i+1} принадлежат одной дуге, рис. 5, *г*) переход к шагу 3.

Шаг 6. Конец работы алгоритма.

Очевидно, что координаты угловых точек при получении массивов ДНП контура выпуклой оболочки определяются точно, в соответствии с их значениями во множестве контактных точек. Координаты же точек сопряжения могут находиться приближенно. Действительно, вследствие дискретного движения

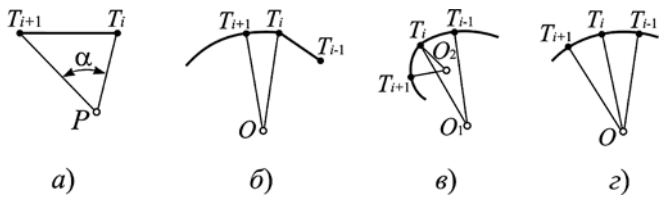


Рис. 5. Взаимное расположение контактных точек

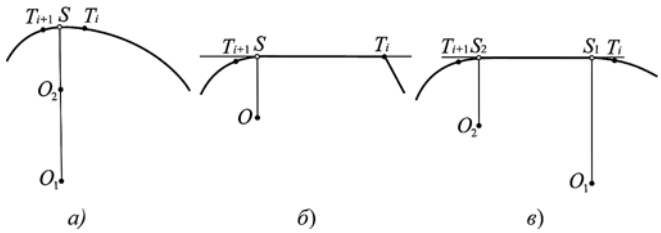


Рис. 6. Точки сопряжения в контуре выпуклой оболочки

ОП по дуге контактные точки не всегда в точности совпадают с точками сопряжения. Например, точка S (рис. 6, *а*) сопряжения дуг с центрами в точках O_1 и O_2, и сопряжения дуги и отрезка (рис. 6, *б*) может находиться между точками T_i и T_{i+1}. Также точки сопряжения S_1 и S_2 (рис. 6, *в*) могут не принадлежать множеству контактных точек.

Положение точек сопряжения может быть найдено с точностью, с которой определены координаты центров дуг, точность которых, в свою очередь, обусловливается точностью нахождения координат контактных точек, принадлежащих дугам.

Искомые точки сопряжения находятся следующим образом.

Сопряжение дуга—дуга: на прямой, проходящей через центры дуг O_1 и O_2, в направлении от большего радиуса к меньшему от одного из центров откладывается соответствующий этому центру радиус и находится точка сопряжения S (рис. 6, *а*).

Сопряжение прямая—дуга (прямая проходит через угловую точку T_i): к дуге проводится касательная, проходящая через эту угловую точку, и определяется точка сопряжения S (рис. 6, *б*).

Сопряжение дуга—прямая—дуга (имеются точка T_i одной дуги и точка T_{i+1} другой дуги): к окружностям с центрами в точках O_1 и O_2, проводится касательная и тем самым определяются точки сопряжения S_1 и S_2 (рис. 6, *в*).

При назначении шага $\Delta\theta$ вращения ОП должны выполняться следующие требования. Во-первых, он должен быть меньше разности углов касательных в угловых точках слева и справа от них (в противном случае эти угловые точки могут не попасть во множество контактных точек, что приведет к искажению контура выпуклой оболочки). Во-вторых, он должен быть таким, чтобы угол α (см. рис. 5, *а*), соответствующий отрезку контура заданной фигуры, имеющему минимальную длину, был меньше этого шага.

Учет допусков на размеры фигуры при ее аппроксимации

Рассмотренные выше алгоритмы аппроксимации используют точечные значения размеров исходной фигуры. Однако в конструкторско-технологическом проектировании размеры деталей задаются с допусками, т. е. интервалами. Изменение значений размеров в поле допуска приводит не только к изменению пло-

щади аппроксимирующих фигур. Оно может приводить и к изменению схемы замыкания деталей (если рассматривается фигура, состоящая из группы контуров), что в свою очередь может вызвать существенное изменение конфигурации аппроксимирующих фигур. Как следствие, возникает задача учета допусков на размеры при решении задачи размещения деталей, определяемых плоскими контурами, в заданной области.

Для решения данной задачи необходимо определить такие значения размеров, лежащие в поле допуска, при которых значение площади исходной фигуры будет максимальным, т. е. найти ее верхнюю границу. В этом случае будет максимальной и площадь АФ, и при решении задачи ее размещения в заданной области можно гарантированно утверждать, что найденное решение будет справедливо для всего множества сочетаний значений размеров фигуры, лежащих в поле их допуска.

Рассмотрим схему получения значений размеров исходной фигуры, дающих ее максимальную площадь. Обозначим верхнее и нижнее отклонения размера P_i как $\Delta^{\text{В}}_{P_i}$ и $\Delta^{\text{Н}}_{P_i}$ соответственно. Тогда его значение будет находиться в пределах: $P_{i\text{ном}} + \Delta^{\text{Н}}_{P_i} \leqslant P_i \leqslant P_{i\text{ном}} + \Delta^{\text{В}}_{P_i}$, или, как принято записывать в конструкторско-технологических документах, $P_i = P_{i\text{ном}}{}^{+\Delta^{\text{В}}_{P_i}}_{-\Delta^{\text{Н}}_{P_i}}$ (или $P_i = P_{i\text{ном}} \pm \Delta_{P_i}$, если отклонения симметричны), где $P_{i\text{ном}}$ — номинальное значение размера P_i. Если известны передаточные отношения ξ_i, ($\xi_i = \pm 1$), показывающие влияние размеров фигуры на ее площадь (при $\xi_i = 1$ размер P_i увеличивает площадь, при $\xi_i = -1$ уменьшает), то совокупность размеров фигуры, дающая максимальную площадь фигуры, определится следующим выражением:

$$P'_i = \begin{cases} P_{i\text{ном}} + \Delta^{\text{В}}_{P_i}, \text{ если } \xi_i = 1; \\ P_{i\text{ном}} + \Delta^{\text{Н}}_{P_i}, \text{ если } \xi_i = -1. \end{cases} \quad (3)$$

Нахождение передаточных отношений легко определяется вручную для достаточно простых исходных фигур. Для сложных фигур не всегда очевидно влияние размеров на их площадь. В этом случае автором рекомендуется использовать систему автоматизированного расчета массово-геометрических характеристик объектов (САР МГХ) [12] (кроме определения передаточных отношений по этой программной системе можно вычислять центр тяжести фигуры, принимаемый за полюс, и ее площадь). Здесь следует отметить то обстоятельство, что САР МГХ использует в качестве входных данных уже имеющееся описание фигуры на языке ПЛОГАС.

После определения размеров по выражению (3) повторяются расчет опорной функции $h(\theta)$ и вычисление множества контактных точек. Далее по приведенным выше алгоритмам находятся искомые АФ с учетом допусков на размеры исходной фигуры.

Проведенные автором расчеты с использованием САР МГХ при различных размерах и зависящих от этих размеров допусков (размеры и допуски взяты из табл. 1.59 справочника [13]), показывают, что учет допусков может увеличивать площадь аппроксимирующей фигуры до 30 %.

Пример

В качестве примера рассмотрим фигуру (рис. 7), для которой найдем различные аппроксимирующие ее фигуры.

Пусть размеры фигуры, заданные в миллиметрах, будут такими (допуски взяты из табл. 1.59 справочника [13]): $P_1 = 7 \pm 1$; $P_2 = 33 \pm 1{,}5$; $P_3 = 8 \pm 1$; $P_4 = 22 \pm 1$; $P_5 = = 11 \pm 1$; $P_6 = 5 \pm 0{,}5$; $P_7 = 20 \pm 1$. С помощью

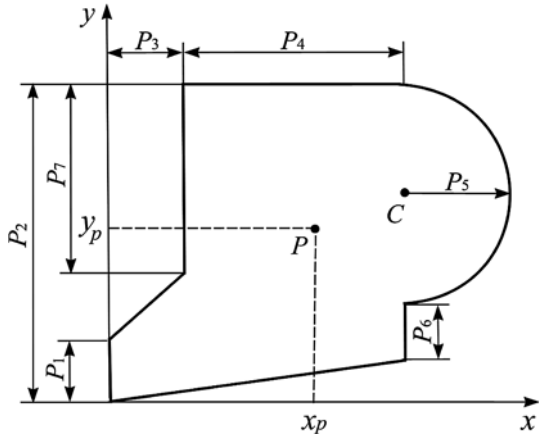


Рис. 7. Исходная фигура:
точка P — полюс фигуры; точка C — центр дуги окружности

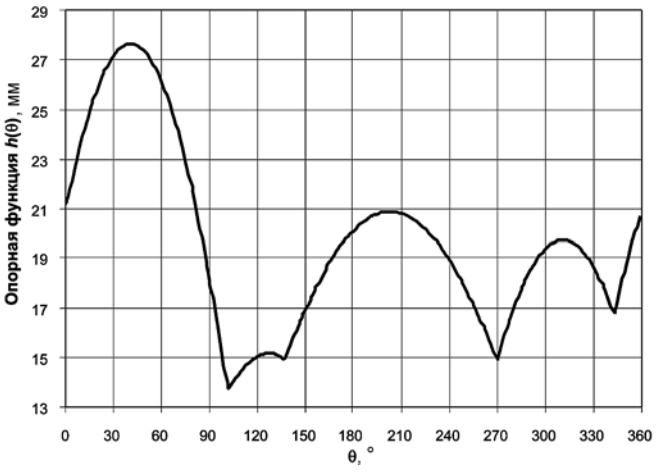


Рис. 8. Опорная функция $h(\theta)$ исходной фигуры

Таблица 1

Аппроксимирующие фигуры

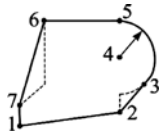
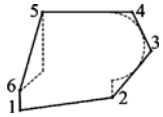
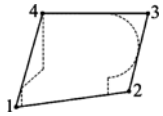
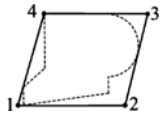
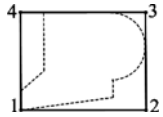
№ фигуры	Аппроксимирующие фигуры
1	Выпуклая оболочка $F = 1001$ ($F' = 1149$) 
2	Шестиугольник $F = 1019$ 
3	Трапеция $F = 1095$ 
4	Параллелограмм $F = 1217$ 
5	Прямоугольник $F = 1353$ 

Таблица 2

Массивы ДНП контура аппроксимирующих фигур

№ фигуры	Координаты точек	Номера характерных точек АФ						
		1	2	3	4	5	6	7
1	x	0	30	38,045	30	30,096	8	0
	y	0	6	14,498	22	33	33	7
1′	x	0	32	40,776	32	32,096	9	0
	y	0	5	14,316	22,5	34,500	34,5	8
2	x	0	30	43,065	37,1	33	8	—
	y	0	6	19,8	33	8	13	—
3	x	−2,295	35,610	46,655	8	—	—	—
	y	−0,459	7,122	33	33	—	—	—
4	x	−2,154	34,72	44,894	8	—	—	—
	y	0	0	33	33	—	—	—
5	x	0	41	41	0	—	—	—
	y	0	0	33	33	—	—	—

Таблица 3

Пересчет номинальных значений размеров исходной фигуры

Параметры пересчета	№ размера						
	1	2	3	4	5	6	7
ξ_i	1	1	1	1	1	1	−1
$P_{i\text{ном}}$, мм	7	33	8	22	11	5	20
Отклонение, мм	1	1	1	1	1	0,5	−1
P_i', мм	8	34	9	23	12	5,5	19

САР МГХ [12] вычислены координаты центра тяжести фигуры (полюс фигуры) и ее площадь при номинальных значениях размеров: $x_p = 20{,}9$; $y_p = 18{,}1$, $F = 906$. Шаг угла вращения ОП $\Delta\theta = 0{,}5°$. Длина ОП принята равной двойному значению бóльшего габаритного размера фигуры.

На основе полученных методом стыковки расчетов построена опорная функция (рис. 8) и вычислены координаты контактных точек.

По приведенным выше алгоритмам найдены площади F некоторых аппроксимирующих фигур, полученных при номинальных значениях размеров (табл. 1) и массивы ДНП их контуров (табл. 2). Кроме того, для выпуклой оболочки эти значения определены при размерах исходной фигуры с учетом допусков (то же самое можно найти и для остальных АФ, приведенных в табл. 1). В табл. 3 показан пересчет значений номинальных размеров исходной фигуры, дающих максимальную площадь выпуклой оболочки; в табл. 1 эта площадь обозначена как F', а табл. 2 содержит массивы ДНП контура этой оболочки — номер фигуры 1′.

Аппроксимирующие фигуры в табл. 1 расположены в порядке возрастания их площадей. Отметим, что для выпуклой оболочки площадь F' на 15 % больше площади F.

Заключение

На основе метода стыковки предложены оригинальные алгоритмы аппроксимации, позволяющие для фигур сложной геометрической формы получать достаточно широкий спектр аппроксимирующих фигур: простые фигуры и выпуклая оболочка (к простым фигурам относятся выпуклые n-угольники и окружность). Выходной информацией алгоритмов является дискретно-непрерывное представление контура аппроксимирующих фигур. Такое представление позволяет строить простые и универсальные алгоритмы работы с аппроксимирующими фигурами.

На основе представленных в статье алгоритмов разработана программа, позволяющая находить:

1) по значениям опорной функции среди простых фигур фигуру, имеющую минимальную площадь при заданном ее виде или среди всех простых фигур;

2) по значениям множества контактных точек выпуклую оболочку при номинальных значениях размеров заданной фигуры.

Рассмотрен способ учета допусков на размеры исходной фигуры.

Приведен пример, в котором для заданной фигуры находятся некоторые аппроксимирующие фигуры.

Отличительной особенностью предлагаемой технологии решения задач аппроксимации является ее универсальность по отношению к форме заданных фигур и к форме аппроксимирующих фигур. Кроме того, данная технология позволяет значительно сократить время подготовки исходных данных (за счет параметрического описания геометрии заданных фигур) и повысить точность получения результатов аппроксимации, зависящую от шага вращения опорной прямой.

Результаты, изложенные в статье, могут использоваться во многих приложениях, где требуется решать задачи размещения, а именно: машинная графика, распознавание образов и изображений, автоматизация проектирования, технологические процессы.

Список литературы

1. **Петунин А. А., Мухачева Э. А., Филиппова А. С.** Метод прямоугольной аппроксимации для решения задач нерегулярного раскроя-упаковки технологий // Информационные технологии. 2008. № 1. С. 28—31.

2. **Белякова Л. Б., Горбунов В. Н.** Определение прямоугольника наименьшей площади, описанного около заданной фигуры // Всесоюзный межвузовский симпозиум по прикладной математике и кибернетике. Тезисы докладов. Горький, 1967. С. 23—31.

3. **Freeman H., Shapira R.** Determining the minimal-area encasing rectangle for an arbitrary closed curve // Comm. ACM. 1975. V. 18, N. 7. P. 409—413.

4. **Мартынов В. В.** Аппроксимация плоского контура многоугольником при решении задач оптимизации размещения деталей обводообразующей оснастки // Начертательная геометрия и машинная графика в практике решения инженерных задач. Омск: ОмПИ, 1988. С. 73—78.

5. **Стоян Ю. Г., Гиль Н. И.** Методы и алгоритмы размещения плоских геометрических объектов. Киев: Наук. Думка, 1976. 247 с.

6. **Глушко А. Г., Рвачев В. Л.** Об одной задаче оптимального раскроя // Кибернетика. 1967. № 1. С. 92—95.

7. **Горанский Г. К., Горелик А. Г., Зозулевич Д. М., Трайнев В. А.** Элементы теории автоматизации машиностроительного проектирования с помощью вычислительной техники. — Минск: Наука и техника, 1970. 336 с.

8. **Бляшке В.** Круг и шар. М.: Наука, 1967. 232 с.

9. **Стоян Ю. Г.** Размещение геометрических объектов. Киев: Наук. Думка, 1975. 239 с.

10. **Аввакумов В. Д.** Определение взаимного положения объектов при их перемещении в плоскости // Информационные технологии. 2006. № 10. С. 52—58.

11. **Аввакумов В. Д.** Функциональная размерная параметризация в САПР // Известия Тульского государственного университета. Сер. Машиностроение. Вып. 6 (специальный). Сборник избранных тр. конф. "Автоматизация и информатизация в машиностроении 2000" (АИМ 2000). 2000. С. 22—28.

12. **Аввакумов В. Д.** Система автоматизированного расчета массово-геометрических характеристик объектов // Сборка в машиностроении, приборостроении. 2008. № 11. С. 38—42.

13. **Палей М. А., Романов А. Б., Брагинский В. А.** Допуски и посадки: Справочник. В 2-х ч. Ч. 1 (8-е изд., перераб. и доп.). СПб.: Политехника, 2001. 576 с.

V. D. Avvakumov, Associate Professor, e-mail: awwam@mail.ru,
Novouralsk Technological Institute of the National Research Nuclear University MEPhI

Approximation Figures of Complex Geometric Shapes Using the Docking

We consider the problem of constructing approximate figures (simple shapes and the convex hull) for figures of complex geometric shapes. Simple shapes are convex n-gon and the circle; both convex hull and complex shapes can be composed of line segments and circular arcs.

The need for approximation arises in solving problems with geometric shapes figures organize complex shape figures (nesting problem): for simple shapes it is much faster to check the conditions of their non intersection. Constructing convex hulls is a very important procedure in many applications not only in itself, it can be part of other algorithms using this procedure as a component.

The first section contains the definitions used in this article: the well-known and introduced by the author. The basic concept is introduced among the arrays of discrete-continuous representation of the sidebar containing the coordinates of the connection points of contour elements and, if the contour has the arcs of their centers.

The second section describes the methodological basis for solutions of approximation algorithms and the method of forming the source data. Methodological basis for the proposed algorithms is a method of docking developed by the author. This method allows to find data for the algorithms constructing desired shapes: a discrete representation of the supporting function for the construction of simple shapes and a plurality of contact points of the of supporting line with the figure for finding the convex hull.

The third section is devoted to the construction of simple approximate figures. In general, for finding an approximating n-gon two procedures (primary and secondary) are used. The first (coarse) approximation of the n-gon is determined by the initial procedure. The application of secondary procedure allows an exact solution.

A plurality of contact points, obtained by joining is a discrete representation of the convex hull because all contact points got in accordance with the algorithm are those of the contour. However, this set contains a significant amount of redundant information. The fourth section describes an algorithm for obtaining arrays of discrete-continuous contour representation of the convex hull based on a plurality of contact points.

Method of dimensional tolerances application of the original figure with its approximation is presented in the fifth section. The substantiation of the need to consider the tolerances in solving problems of placing objects on the plane is given. Calculations show that the inclusion of tolerance can increase the area of approximating figures by up to 30 %.

The results of algorithms work for obtaining approximating figures are arrays of discrete-continuous representation of the contour. Such representation of the contour allows to build simple and universal algorithms for flat figures in solving location problems.

There is an example in which the figures are given for some approximate figures.

The results presented in this article may be used in applications where it is required to solve the problem of locating geometric objects: computer graphics, pattern recognition and image processing, computer-aided design and manufacturing processes.

Keywords: *approximation, the figure of complex geometric shapes, simple figures, convex hull, contour, algorithms, continuous-discrete representation of the contour, nesting, supporting line, tolerance on size*

References

1. **Petunin A. A., Muhacheva E. A., Filippova A. S.** Metod prjamougol'noj approksimacii dlja reshenija zadach nereguljarnogo raskroja-upakovki tehnologij. *Informacionnye tehnologii*. 2008. N. 1. P. 28—31.

2. **Beljakova L. B., Gorbunov V. N.** Opredelenie prjamougol'nika naimen'shej ploshhadi, opisannogo okolo zadannoj figury. *Vsesojuznyj mezhvuzovskij simpozium po prikladnoj matematike i kibernetike. Tezisy dokladov*. Gor'kij, 1967. P. 23—31.

3. **Freeman H., Shapira R.** Determining the minimal-area encasing rectangle for an arbitrary closed curve. *Comm. ACM*. 1975. V. 18, N. 7. P. 409—413.

4. **Martynov V. V.** Approksimacija ploskogo kontura mnogougol'nikom pri reshenii zadach optimizacii razmeshhenija detalej obvodoobrazujushhej osnastki. *Nachertatel'naja geometrija i mashinnaja grafika v praktike reshenija inzhenernyh zadach*. Omsk: OmPI, 1988. P. 73—78.

5. **Stojan Ju. G., Gil' N. I.** *Metody i algoritmy razmeshhenija ploskih geometricheskih ob'ektov*. Kiev: Nauk. Dumka, 1976. 247 p.

6. **Glushko A. G., Rvachev V. L.** Ob odnoj zadache optimal'nogo raskroja. *Kibernetika*. 1967. N. 1. P. 92—95.

7. **Goranskij G. K., Gorelik A. G., Zozulevich D. M., Trajnev V. A.** *Jelementy teorii avtomatizacii mashinostroitel'nogo proektirovanija s pomoshh'ju vychislitel'noj tehniki*. Minsk: Nauka i tehnika, 1970. 336 p.

8. **Bljashke V.** *Krug i shar*. M.: Nauka, 1967. 232 p.

9. **Stojan Ju. G.** *Razmeshhenie geometricheskih ob'ektov*. Kiev: Nauk. Dumka, 1975. 239 p.

10. **Avvakumov V. D.** Opredelenie vzaimnogo polozhenija ob'ektov pri ih peremeshhenii v ploskosti. *Informacionnye tehnologii*. 2006. N. 10. P. 52—58.

11. **Avvakumov V. D.** Funkcional'naja razmernaja parametrizacija v SAPR. *Izvestija Tul'skogo gosudarstvennogo universiteta. Serija Mashinostroenie. Sbornik izbrannyh trudov konferencii "Avtomatizacija i informatizacija v mashinostroenii 2000" (AIM 2000)*. 2000. Vypusk 6 (special'nyj). P. 22—28.

12. **Avvakumov V. D.** Sistema avtomatizirovannogo rascheta massovo-geometricheskih harakteristik ob'ektov. *Sborka v mashinostroenii, priborostroenii*. 2008. N. 11. P. 38—42.

13. **Palej M. A., Romanov A. B., Braginskij V. A.** *Dopuski i posadki: Spravochnik*. V. 2 ch. Ch. 1 (8-e izd., pererab. i dop.). SPb.: Politehnika, 2001. 576 p.

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4

Дизайнер *Т. Н. Погорелова*. Технический редактор *Е. М. Патрушева*. Корректор *Т. В. Пчелкина*

Сдано в набор 07.07.2014 г. Подписано в печать 19.08.2014 г. Формат 60×88 1/8. Заказ PI914
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д. 19, стр. 1.