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 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.

УДК 004.4'236

А. Р. Шакуров, стажер-исследователь, Национальный исследовательский университет — Высшая школа экономики (г. Москва),
e-mail: amir-shak@yandex.ru

Компонентная технология программирования с поддержкой динамического создания компонентов

Рассматривается функциональная особенность, не имеющая полноценной реализации в существующих технологиях компонентного программирования, которая заключается в возможности динамического создания новых типов данных. Предлагается архитектура программной реализации компонентной модели, поддерживающей данную операцию.

Ключевые слова: *компонент, архитектура, повторное использование, метапрограммирование*

Современные компонентные технологии

Идее компонентной разработки ПО, преследующей цели повторного использования кода и модуляризации структуры программных систем, уже более 50 лет [7]. В настоящее время компонентные технологии успешно используются с различными целями. Несмотря на то, что такие технологии уступают по популярности технологиям, которые положены в основу традиционных языков программирования, они столь же разнообразны по дизайну и назначению. По этой причине в данной статье кратко рассматриваются свойства ряда актуальных технологий и анализируются их общие черты и присущие им ограничения. Предлагается модификация компонентной парадигмы, а также компонентная модель, разработанная с учетом этих предложений. В заключение остановимся на внутренней архитектуре программной реализации данной модели, а также на ее практических применениях.

Компонентные и объектные технологии. Рассмотрим некоторые вводные соображения, связанные с терминологией и с тем, как в понимании автора соотносятся компонентные технологии с объектными[1]. Подобно тому, как класс в объектно-ориентированных языках программирования (ОО ЯП) содержит метаданные объектов данного класса, которые получаются в результате его инстанциирования, компонент в компонентных технологиях (КТ) позволяет создавать и описывает свои экземпляры. Из соображений краткости будет также называть их объектами там, где это не приводит к неоднозначности. Аналогия отношений "класс — объект" и "компонент — экземпляр" подкрепляется еще тем обстоятельством, что экземпляр, как и объект, представляет собой "черный ящик", т.е. сущность времени исполнения, которая скрывает собственные данные и функциональные возможности (поведение) за строго определенным интерфейсом, посредством которого и осуществляется всякий доступ к этой сущности. Описанные сходства не означают, однако, что всякая объектная технология является и компонентной или что классы можно считать компонентами. Причина в том, что защита от прямого воздействия на внутреннее состояние объекта (инкапсуляция) еще не определяет независимый характер его функционирования. Это подводит нас к следующим двум ключевым отличиям технологий компонентных от объектных.

- Компонент обладает большей независимостью, чем класс: его можно установить, развернуть и исполнить (создать экземпляр, функциональностью которого можно воспользоваться) независимо от наличия в системе экземпляров других компонентов [11].
- Сочленение экземпляров друг с другом (в целях их композиции в готовое приложение) осуществляется стандартным образом и является достаточно простой операцией. В противоположность этому, соединение объектов в готовую объектно-ориентированную

[1] В данной работе термины "объектный" и "объектно-ориентированный" являются синонимами.

систему — задача непростая. Этот аспект иногда формулируют как "программирование без написания кода", однако такая формулировка представляется неоднозначной и потому неудачной. Следует отличать императивный код, содержащий пошаговые инструкции того или иного алгоритма, от кода декларативного как средства спецификации намерений программиста. Императивный код действительно выходит за рамки компонентной идеологии. Вместе с тем код декларативный может по-прежнему быть удобным средством взаимодействия разработчика с платформой компонентной разработки (наряду с такими средствами, как, например, диаграммное или табличное декларирование требований) [2].

Большое значение также имеет понятие компонентной платформы — среды исполнения, обеспечивающей функционирование и взаимодействие экземпляров и выдвигающей требования к ним. Компонентную платформу также называют компонентной моделью, хотя последним термином часто обозначают и некоторую абстрактную, не имеющую конкретной программной реализации (как правило, специфицированную) платформу.

Существующие компонентные решения. Все разнообразие созданных на настоящее время компонентных технологий можно классифицировать следующим образом:

1) КТ, призванные упростить организацию взаимодействия между различными процессами (возможно, исполняющимися на удаленных компьютерах), по сути, это средства межпроцессного взаимодействия (IPC) и/или удаленного вызова процедур (RPC), выполненные в компонентном виде, например, IBM SOM/DSOM[2], Microsoft COM/DCOM [9], OMG CORBA (после включения компонентной модели CCM в состав стандарта CORBA 3.0) [8];

2) КТ общего назначения, призванные упростить разработку ПО широкого спектра, включая проект Ptolemy[3], JavaBeans [10], компонентную модель[4].NET, Fractal [4], ArchJava [3] и др.;

3) технологии, упрощающие разработку приложений определенной предметной области, либо создание моделей, учитывающих специфику некоторой предметной области, например, OmNet++ (моделирование сетей связи)[5], VRML [5] и результат его дальнейшего развития X3D [6] (3D-моделирование), WireFusion (3D-моделирование с фокусом на создание интерактивных сцен на основе VRML- и X3D-компонентов)[6].

Разумеется, классификация такого числа разнородных проектов не может быть строгой. Так, JavaBeans, будучи технологией общего назначения, была создана в первую очередь для визуального проектирования графических пользовательских интерфейсов. Модель Enterprise JavaBeans, созданную для разработки серверной части корпоративных приложений, трудно отнести как к технологиям общего назначения, так и к технологиям, учитывающим специфику определенной предметной области. Проект Ptolemy позволяет создавать приложения различного рода, но основной акцент сделан на параллелизм и создание встраиваемых систем. Тем не менее, приведенная классификация оказывается полезной в попытках понять причины, определяющие те или иные функциональные и архитектурные особенности КТ. Так, технологии, отнесенные к первой группе, призваны обеспечить взаимодействие между приложениями, разработанными с использованием различных языков программирования. По этой причине модели, положенные в их основу, наиболее близки к моделям соответствующих языков. Технологии, подобные COM, например, оперируют объектами, каждый из которых принадлежит к определенному классу и реализует один или более интерфейсов. Последние представляют собой совокупность методов. Таким образом, используемая COM модель очень напоминает объектные модели таких ОО ЯП как C# или Java. Однако при этом следует заметить, что указанное сходство не является полным. Упомянутые языки интенсивно используют наследование, тогда как COM избегает этого, чтобы не столкнуться с так называемой "проблемой ломкого базового класса" (*fragile base class problem*), заключающейся в том, что изменения базового класса в глубокой иерархии наследования приводят к нарушению бинарной совместимости со всеми его подклассами.

Модели, лежащие в основе КТ общего назначения, не будучи столь тесно связаны с языками программирования, отличаются бóльшим разнообразием. Так, интерфейс экземпляров компонентов модели Ptolemy II, называемых áкторами, представляет собой совокупность параметров, используемых для настройки поведения актора, и портов. Между портами создаются каналы сообщений, благодаря чему акторы могут взаимодействовать. Составленная таким образом система имеет собственный интерфейс, также представленный параметрами и портами. Последние, в свою очередь, соединяются с портами акторов, составляющих систему, а также с портами других систем. Иными словами, имеет место иерархическая структура. Примечательной особенностью модели Ptolemy II является то, что структурам акторов не приписывается семантическая нагрузка. Семантику привносит "режиссер" модели. Эта специальная сущность определяет модель вычислений, которая задает, каким образом осуществляется взаимодействие между акторами.

[2] ftp://ftp.software.ibm.com/publications/clubod/som30/index.html
[3] http://ptolemy.berkeley.edu/ptolemyII/
[4] Компонентная модель, поддерживаемая платформой.NET, не имеет собственного названия. Речь идет о модели, которую определяют классы и интерфейсы, расположенные в пространстве имен System.ComponentModel. http://msdn.microsoft.com/netframework/
[5] http://www.omnetpp.org/
[6] http://www.demicron.com/wirefusion/

Во многом схожие компонентные модели JavaBeans и.NET имеют больше общего с ОО ЯП. В них интерфейс экземпляра представлен свойствами, методами и событиями. События возникают в ответ на изменения, происходящие с объектом, что приводит к выполнению соответствующего кода обработки события. Заметим, что необходимость писать код, а также отсутствие возможности иерархически "вкладывать" компоненты друг в друга подразумевают, что продуктивное использование компонентных моделей JavaBeans и.NET возможно только в тесной связке с соответствующими языками программирования (Java и C#) и инструментариями разработки.

Еще одним примером может служить Fractal. Спецификация этой компонентной модели разделена на несколько частей, названных уровнями контроля. Каждая из этих частей описывает ту часть API, которая может быть использована для доступа к возможностям, предоставляемым компонентом, поддерживающим соответствующий уровень контроля. Экземпляр компонента Fractal самого нижнего уровня контроля есть не что иное, как обыкновенный объект (в смысле ОО ЯП), взаимодействие с которым осуществляется посредством вызова методов. Это обстоятельство делает возможной интеграцию приложений, разработанных с помощью Fractal, с обычными, некомпонентными, разработками. Следующий уровень *внешней интроспекции* специфицирует, каким образом могут быть обнаружены все интерфейсы экземпляра. Предусматриваются интерфейсы двух видов — серверные (предоставляемые) и клиентские (требуемые). *Уровень конфигурации* позволяет обнаруживать и изменять содержимое экземпляров. Экземпляр состоит из других экземпляров, соединенных друг с другом связями между серверными и клиентскими интерфейсами, и так далее — до примитивных (нижнего уровня) экземпляров. Управление множеством подэкземпляров и связями между ними и осуществляется на данном уровне. Последующие уровни контроля отвечают за конструирование экземпляров и их типизацию.

Технологии, ориентированные на конкретную предметную область, представляют в контексте настоящего исследования наименьший интерес, так как его цель — создание технологии общего назначения, подходящей для разработки приложений широкого спектра. Рассмотрим только одну из них — VRML/X3D. Во-первых, она не привязана к конкретной среде разработки (как, например, WireFusion) и, во-вторых, имеет модель, допускающую распространение на более широкий спектр задач, нежели 3D-моделирование.

Модель VRML представляет собой граф узлов, описывающих объекты моделируемой сцены. Каждый узел характеризуется некоторым типом. При этом узлы, задающие аудио-визуальное представление того или иного объекта, объединены в иерархию. Например, узел типа Appearance, определяющий визуальные свойства объекта, может содержать узел типа Material, который, в свою очередь, содержит узел типа Color, задающий цвет материала объекта. Тип узла характеризуется именем, полями, перечнями принимаемых и генерируемых событий, а также реализацией, определяющей аудио-визуальное представление узла (если таковое имеется), механизмы генерирования событий, а также реакцию узла на принимаемые события. Между генерируемыми и принимаемыми событиями узлов графа сцены осуществляется маршрутизация событий: первые возникают в ответ на какие-либо изменения во внешней среде (например, на срабатывание таймера) или действия пользователя и попадают во вторые, что вызывает изменения в состоянии узла в ответ на произошедшее событие.

Модель VRML предусматривает возможность определения пользовательских типов узлов в дополнение к стандартным. Инструкция PROTO позволяет задать реализацию нового типа на основе существующих, дополнить ее информацией об интерфейсе (полями и событиями, связанными с реализационной частью) и получить, тем самым, новый тип узла VRML.

По причинам, представленным ниже, в контексте настоящего исследования большое значение придано операции определения новых типов данных (применительно к КТ). Рассмотрим ее подробнее.

Динамическое создание типов данных. Операция динамического определения новых типов данных на основе существующих (*runtime type definition*, RTTD) делает КТ саморасширяемой и потому более универсальной подобно тому, как возможность определять новые классы в ОО ЯП позволяет более естественным образом создавать модели, адекватно отражающие ту или иную предметную область. Невозможно представить объектную технологию общего назначения, не позволяющую пользователю расширять существующую систему типов. В связи с этим автор считает, что единственной причиной, по которой компонентные технологии сегодня не предоставляют полноценной поддержки RTTD, является их зависимость от сторонних (выходящих за рамки компонентной модели) технологий. Примером может служить упомянутая выше комбинация JavaBeans и Java.

С учетом изложенных соображений, КТ, поддерживающая RTTD, имеет возможность стать технологией общего назначения, позволяющей решать широкий спектр практических задач, не выходя за ее рамки. Рассмотрим, каким образом может быть реализована данная функциональная особенность. Существует несколько различных способов поддержки динамического определения новых типов данных:

- генерация исходного кода с последующим вызовом компилятора либо непосредственная генерация бинарного или байт-кода;
- использование прототипов, когда вместо создания нового типа в системе сохраняется настроенный нужным образом объект-прототип, а процедура инстанциирования типа заменяется его клонированием;

- применение технологии отражения и метапрограммирования времени исполнения, если средства реализации КТ располагают такими возможностями;
- непосредственная поддержка со стороны компонентной модели.

Отметим, что под механизмом отражения понимается возможность получения информации о системе типов во время исполнения и манипулирования объектами на основе этой информации. Метапрограммированием называется модификация текущей системы типов, также во время исполнения. Там, где это не порождает неоднозначности, будем использовать термин "отражение" для обозначения обоих этих понятий.

Первое из приведенных решений применимо далеко не всегда, так как компилятор или средства генерации байт-кода доступны не во всякой среде исполнения. Это ограничение, в частности, свойственно встраиваемым системам.

Использование прототипов также сопряжено с рядом трудностей. Во-первых, система контроля типов, если таковая есть, становится практически бесполезной. Так, если два прототипа представляют собой объекты одного класса, настроенные различным образом, то для того чтобы понять, клонированием которого из прототипов был создан тот или иной объект (если это вообще возможно), потребуется изучить текущее состояние объекта. Во-вторых, создание прототипа (настройка объекта под контекст использования) не изменяет метаинформации, хранящейся в его классе, что порождает ряд проблем с эффективностью. Более того, если класс рассматриваемого объекта таков, что внутреннее состояние его экземпляров изменяется со временем, применение прототипирования недопустимо, так как каждое клонирование будет порождать объект, отличающийся от своих "собратьев".

Использование метапрограммирования времени исполнения, поддерживаемого современными языками программирования, такими как Python или Ruby, требует существенных усилий со стороны разработчика, поскольку эти возможности выходят за рамки повседневного использования данных языков, сложны в применении и требуют написания значительных объемов кода. Сложность применения рассматриваемых механизмов, впрочем, является следствием сложности языков и структур, описываемых ими. Спектр изменений, вносимых в функциональность класса при порождении из него нового, в ОО ЯП поистине безграничен: от добавления новой переменной до изменения интерфейса или включения нового класса в перечень суперклассов. Такое разнообразие модификаций и приводит к сложности их реализации.

Компонентная модель, оперирующая меньшим числом понятий, делает операцию создания типа достаточно простой и формальной с тем, чтобы она не требовала ручного кодирования и умещалась, таким образом, в рамки самой компонентной модели. Итак, будучи упрощенным аналогом механизма метапрограммирования времени исполнения, непосредственная поддержка RTTD со стороны компонентной модели лишена недостатков альтернативных реализаций данной возможности.

Существующие КТ, однако, не предоставляют полноценной поддержки RTTD. Если придерживаться представленной выше классификации, то интегрирующие компонентные технологии меньше других соотносятся с RTTD. Это легко объяснить тем, что они рассчитаны на применение в связке с традиционными инструментами разработки (такими как компиляторы), с помощью которых и осуществляется определение новых типов. То же можно сказать о некоторых КТ общего назначения (JavaBeans и .NET). В других технологиях поддержка RTTD более проработана. Так, Ptolemy II предоставляет возможность превращения модели или актора в класс. Пользователь может создать экземпляр того или иного типа библиотечного компонента, подстроить значения его параметров, после чего конвертировать экземпляр в новый класс, при инстанциировании которого экземпляры изначально будут иметь нужные параметры. По сути, это именно RTTD, но возможности пользовательской "подстройки" типа здесь весьма ограничены. Можно лишь задать структуру составного компонента и значения параметров компонента (как примитивного, так и составного). Вместе с тем диапазон типов этих значений чрезвычайно узок (во всяком случае, так это реализовано в Ptolemy II): фактически, параметры могут быть числового или строкового типа.

Другим примером служит модель Fractal, допускающая создание новых типов интерфейсов, а из них — новых компонентов. Однако функциональные возможности, скрывающиеся за тем или иным типом интерфейса, по-прежнему реализуются на лежащем в основе модели языке программирования.

Наконец, упомянутый механизм PROTO в VRML также призван поддерживать RTTD. Заметим, однако, что существующие реализации (VRML-браузеры), встречая инструкцию PROTO, по факту не создают новый тип узла, а лишь выполняют текстовую макроподстановку [1].

С учетом изложенных выше соображений, представляет интерес КТ, непосредственно поддерживающая создание новых компонентов на основе существующих. В результате исследований, направленных на создание такой технологии, была разработана и реализована компонентная модель, которая рассматривается далее.

Модель с точки зрения пользователя

Исходя из представленного выше описания свойств КТ, можно выделить следующие их ключевые особенности (принципы).

1. Разрабатываемая программная система представляется в виде сети взаимодействующих экземпляров ее компонентов, что подразумевает типизацию: для каждого экземпляра известно, какой компонент является его типом.

2. Существует четкое разделение интерфейса и реализации экземпляров (инкапсуляция).

3. Интерфейс представляет собой набор свойств (портов, полей и др.), доступных для чтения/записи, и событий.

4. Реализация либо "непрозрачна" (не специфицируется компонентной моделью), либо представляет собой совокупность экземпляров других компонентов.

Отдельные технологии, разумеется, могут отклоняться от перечисленных принципов. Например, JavaBeans не поддерживает иерархическую организацию экземпляров. Детали реализации п. 3 отличаются особым разнообразием. Однако в целом приведенный перечень описывает то, что на настоящее время, по мнению автора, принято считать КТ.

Экземпляры. Будем придерживаться перечисленных принципов, поскольку они хорошо знакомы пользователям. Экземпляр компонента предлагаемой модели представляет собой сущность времени исполнения, инкапсулирующую некоторые данные и поведение за интерфейсом. Интерфейс состоит из свойств — именованных атрибутов фиксированного типа, характеризующихся текущим значением, к которым могут быть применимы операции чтения, записи и связывания. Возможность применения определяется разрешениями, задающими, какие свойства доступны для той или иной операции. Информацию о разрешениях хранит компонент, являющийся типом рассматриваемого экземпляра. Семантика первых двух операций очевидна — получение и установка значения свойства, соответственно. Смысл операции связывания состоит в следующем: если связать свойство *A* со свойством *B*, то всякое последующее изменение свойства *A* будет приводить к записи нового значения *A* в *B*. Свойства при этом могут принадлежать как одному, так и разным экземплярам, свойство *A* должно быть доступно для связывания, а *B* — для записи. Возможность связывания свойств необходима для эффективной реализации механизма обратных вызовов. В то же время, в модель не вводится отдельное понятие события во избежание чрезмерного ее усложнения.

Что касается реализации экземпляров, то она определяется тем, к какой категории этот экземпляр принадлежит. Существует три категории экземпляров: примитивные, скомпилированные, скомпонованные. Первые — это неизменяемые объекты-значения (например, числа); их реализация есть не что иное, как константная память, хранящая значение. Экземпляры второй категории введены в модель для поддержки сторонних технологий. Их реализация — это экземпляр стороннего компонента в совокупности с информацией о том, как с ним взаимодействовать. Наконец, реализация экземпляров скомпонованных компонентов представляет собой совокупность экземпляров других компонентов, взаимодействующих друг с другом посредством связей между свойствами и с интерфейсом через разделяемые свойства. Механизм разделяемых свойств подобен механизму связывания свойств в том плане, что изменения, происходящие с одним из разделяемых свойств, происходят и со всеми другими. Реализуется он таким образом, что для хранения значений свойств нескольких экземпляров физически используется одна и та же область памяти.

Компоненты. Экземпляры получаются в результате инстанциирования компонентов, хранящихся в специально предназначенной для этого библиотеке типов. Компоненты, будучи типами данных своих экземпляров, содержат метаданные, описывающие структуру интерфейсной и реализационной частей будущих экземпляров.

В соответствии с приведенным выше описанием, метаданные интерфейса содержат дескрипторы свойств, специфицирующих имена свойств, типы их значений, разрешения на чтение, запись и связывание и, возможно, значения по умолчанию. Метаданные реализации различны для скомпилированных и скомпонованных компонентов и примитивных типов. Отметим, что скомпонованный экземпляр и экземпляр скомпонованного компонента — это одно и то же. Примитивные типы не являются компонентами, поскольку не могут быть сконструированы без параметров (будучи неизменяемыми, они требуют указания как минимум инициализирующего значения). Для примитивного типа это сведения об объеме памяти, необходимом для хранения значений его экземпляров. Для скомпилированного компонента здесь хранится информация о том, как получить экземпляр компонента сторонней технологии и связать его с интерфейсной частью. Наконец, скомпонованный компонент содержит сведения об экземплярах, которые должны быть инстанциированы в качестве подэкземпляров будущего объекта, а также об их связях между собой (посредством связанных свойств) и с интерфейсной частью (посредством разделяемых свойств).

Контейнер. Подобно тому, как объекты ОО ЯП "живут" в среде исполнения, экземпляры инстанциируются и взаимодействуют в контексте контейнера. Здесь пользователь может создавать экземпляры любых компонентов из библиотеки типов, просматривать и устанавливать значения их свойств, а также связывать их друг с другом.

Контейнер, однако, имеет еще одно функциональное назначение: он способен реинтерпретировать созданную внутри него структуру объектов как прототип реализационной части будущего скомпонованного компонента. Пользователю остается лишь указать интерфейс нового типа и то, каким образом реализация взаимодействует с ним (иными словами, какие свойства каких подэкземпляров должны быть разделены с какими свойствами, составляющими интерфейс), после чего контейнер создаст новый скомпонованный компонент. В дальнейшем вновь созданный компонент можно будет использовать также как и любой другой.

Рассмотрев, каким образом выглядит предложенная модель с точки зрения пользователя, перейдем к описанию ее реализации.

Архитектура программной реализации модели

Описанная выше модель была реализована на языке Java. Структура реализации в основном отражает модель. Всякий компонент представляется в системе объектом, класс которого является наследником абстрактного класса Type, объявляющим методы для инстанциирования компонента. Аналогично, всякий экземпляр представлен объектом, реализующим интерфейс Instance, методы которого позволяют определять тип данного экземпляра и получать доступ к его свойствам. Компоненты рассматриваются как частный случай экземпляров. Через их свойства доступны все те метаданные, структура которых описана ниже. Типом всякого компонента (как экземпляра) является специальный тип "Тип" (объект класса TypeType), являющийся и своим собственным типом.

Примитивные типы достаточно просты и представляют собой лишь обертку вокруг переменных соответствующих типов. Реализация скомпилированных компонентов определятся тем, какая сторонняя технология положена в их основу. Один такой пример будет рассмотрен далее. Перейдем к анализу структуры скомпонованных компонентов, которые в контексте целей настоящей работы представляют наибольший интерес.

Структура скомпонованного компонента. Класс скомпонованного компонента CompositeType состоит из интерфейсной (CompositeTypeInterfacePart) и реализационной (CompositeTypeImplementationPart) частей (рис. 1). Первая включает в себя произвольное число дескрипторов свойств (класс PropertyDescriptor на диаграмме), каждый из которых задает тип свойства, разрешения доступа (класс Permissions) и, возможно, значение по умолчанию.

Реализационная часть представляет собой совокупность объектов, специфицирующих, какие компоненты должны быть инстанциированы "внутри" будущего экземпляра (объекты класса SubcomponentDescriptor, рис. 2) и каким образом они должны быть взаимосвязаны (объекты класса EventRoute задают событийные связи; начало и конец каждой связи представлены объектами класса SubcomponentPropertyQualifier).

При этом для любого свойства каждого из подкомпонентов может быть указана его настройка для использования в данном контексте (SubcomponentPropertyContextAdjustment), а именно:

- ограничение разрешений доступа (PermissionsModifier): например, доступ к свойству, ранее доступному для записи, можно ограничить доступом только для чтения;
- изменение значения по умолчанию (DefaultValueModifier), которое свойство получит в процессе конструирования; такое изменение может носить различный характер:
  - "вырожденное" изменение: значение остается нетронутым;
  - явно заданное значение;

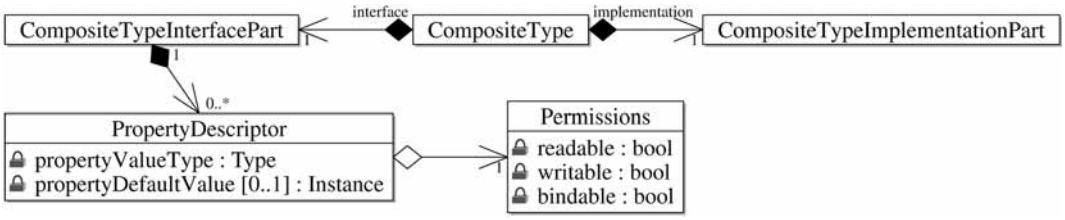


Рис. 1. Структура скомпонованного компонента и его интерфейсной части (в нотации UML). Методы и часть атрибутов не показаны. Доступ к изображенным закрытым атрибутам осуществляется с помощью соответствующих методов

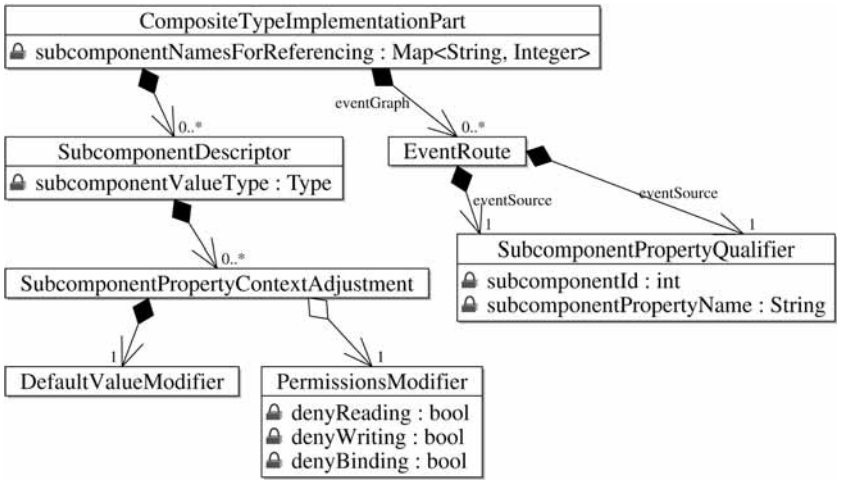


Рис. 2. Реализационная часть скомпонованного компонента

- значение свойства есть экземпляр, доступный в том же контексте (подэкземпляр того же надэкземпляра);
- свойство подэкземпляра должно разделять свойство интерфейса надэкземпляра; при этом изменяется не значение свойства, а само свойство: для хранения его значения не создается новая ячейка памяти (как это происходит в других случаях), вместо этого используется существующая ячейка, соответствующая свойству интерфейса надэкзепляра.

Заметим, что в целях упрощения манипуляции ячейками памяти, выделяемой для хранения значений свойств, и поддержки механизма их связывания была реализована специализированная модель памяти, предоставляющая кроме традиционных констант и переменных связываемые переменные, способные уведомлять подписчиков об изменении хранящихся в них данных. Запросы на выделение памяти осуществляются через единый интерфейс, требующий указания типа памяти (константная, переменная, связываемая). Такой подход позволяет реализовать следующее поведение системы: когда пользователь ограничивает доступ к тому или иному свойству, модель памяти учитывает это, выделяя наиболее "дешевую" (обладающую наименьшими возможностями) память. Например, если свойство, доступное для чтения и записи, было сделано доступным только для чтения, будет выделена константная память (что делает возможным ее совместное использование). Если к свойству запретить доступ для связывания, механизм уведомления подписчиков не будет инициализирован для соответствующей области памяти.

Продемонстрируем, каким образом используется описанная структура скомпонованного компонента в процессе его инстанциирования.

1. Прежде всего, для каждого дескриптора свойства создается ячейка памяти для хранения значений этого свойства. Если инстанциирование происходит в контексте (создаваемый экземпляр будет служить частью реализации другого экземпляра), то вместо создания новой ячейки может быть задействована ячейка свойства надэкземпляра. Затем в свойство записывается значение по умолчанию (явно заданное или получаемое непосредственным интанциированием нужного компонента).

2. Конструируются экземпляры, составляющие реализационную часть будущего объекта. При этом учитывается описанная выше их контекстная настройка, в результате чего для каждого свойства каждого будущего подэкземпляра определяются окончательные разрешения доступа и значение по умолчанию.

3. Между свойствами только что созданных экземпляров устанавливаются событийные связи.

4. Создаются объекты-посредники доступа к свойствам созданного экземпляра.

Последний пункт требует разъяснений. Доступ к свойствам скомпонованного экземпляра осуществляется косвенно, посредством объектов классов PropertyGetter, PropertySetter и PropertyBinder. В процессе инстанциирования они получают ссылки лишь на те свойства, которые им необходимы (объект класса PropertySetter, например, ссылается только на свойства, доступные для записи). Более того, если таких свойств нет, то соответствующий объект-посредник не инициализируется вовсе. Все эти меры делают реализацию более эффективной и позволяют создаваемым экземплярам глубоко подстраиваться под контекст использования.

Описанная внутренняя структура скомпонованного компонента обеспечивает значительную гибкость. Так как типы данных представлены структурами (времени исполнения) обычных объектов (в смысле языка Java), а не скомпилированным кодом, появляется возможность с легкостью манипулировать такими структурами, создавая тем самым новые типы.

Создание новых компонентов. Алгоритм контейнера по созданию нового компонента запускается пользователем, когда тот, будучи удовлетворен поведением созданного прототипа, решает зафиксировать результат в новом типе данных. В процессе работы с контейнером пользователь может не только конструировать в нем экземпляры и манипулировать ими, но и добавлять свойства в интерфейс прототипа, сообщая, тем самым, контейнеру метаданные интерфейса будущего компонента. Важно, что при конструировании очередного экземпляра его свойства могут быть разделены со свойствами интерфейса. Таким образом, разработка интерфейсной части прототипа протекает параллельно с определением его реализации.

Получив команду на создание компонента, контейнер собирает необходимые метаданные, анализируя структуру текущего состояния прототипа. Для каждого его свойства создается дескриптор. Это не вызывает сложностей: тип и права доступа были указаны явно, а в качестве значения по умолчанию принимается значение свойства на момент операции. Для каждого экземпляра внутри контейнера создается дескриптор — подкомпонент, в котором сохраняются его тип, а также данные, относящиеся к его свойствам. Здесь для каждого свойства вычисляется описанный выше объект настройки под контекст использования: ограничения доступа (задаются пользователем явно) и модификатор значения по умолчанию, который получается из анализа структуры прототипа. Например, если обнаруживается, что свойство использует ту же область памяти, что и одно из свойств интерфейса прототипа, фиксируется отношение разделения между ними. Наконец, для каждой связи между свойствами экземпляров создается объект класса EventRoute, фиксирующий ее источник и приемник.

Таким образом, происходит сбор всей необходимой информации, составляющей будущий компонент. Предлагаемый подход продиктован стремлением извлекать как можно больше из структуры прототипа, явно запрашивая только те метаданные, которые невозможно получить из такого анализа. Благодаря этому пользователь может сосредоточиться на работе над прототипом,

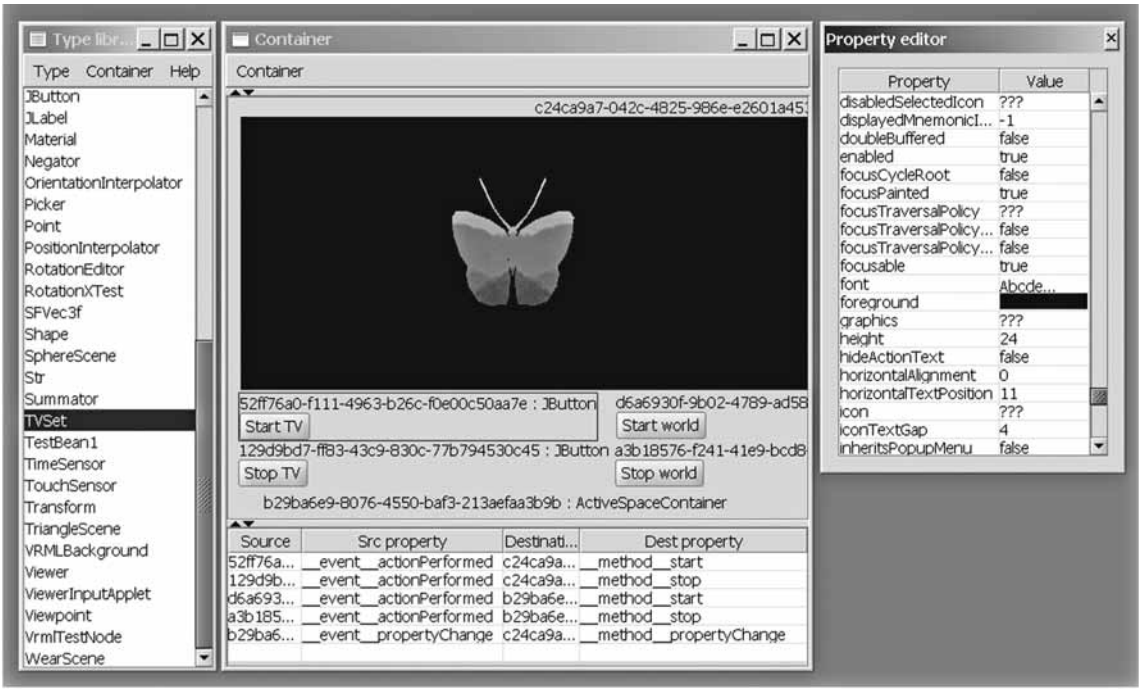


Рис. 3. Пример использования компонентов JavaBeans, реализующих стандартные типы узлов VRML, в качестве скомпилированных компонентов

а не на вводе метаданных. Возможность мгновенно наблюдать поведение создаваемой структуры значительно упрощает разработку[7] и лежит в основе популярности компонентного подхода.

Описанное выше взаимодействие с пользователем, впрочем, относится только к режиму интерактивного проектирования. Модель вполне позволяет осуществлять ввод и из специально разработанного диалекта XML (наподобие того, как это сделано в X3D). В этом случае фокус смещается на описание структуры скомпонованных компонентов, что ближе к декларативному стилю программирования.

Поддержка компонентов JavaBeans. В качестве демонстрации поддержки сторонних технологий в реализацию предложенной модели включена поддержка компонентов JavaBeans в виде скомпилированных компонентов. Последние также имеют свойства для чтения/записи и, кроме того, способны уведомлять об их изменениях событием типа PropertyChangeEvent. Все это однозначно соответствует предложенной модели и потому очевидным образом интегрируется с ней. Существуют, однако, две сложности.

Во-первых, типы свойств компонентов JavaBeans являются обыкновенными классами Java, поэтому в нашей модели соответствующего типа может просто не быть. Можно было бы обернуть произвольный класс либо в "непрозрачный" скомпилированный компонент-оболочку, либо в "прозрачный" компонент, рассматривающий его как компонент JavaBeans. Однако и тот, и другой варианты порождали бы конфликты, вызванные тем, что для некоторого класса существовало бы два различных компонента-обертки. Это неизбежно, поскольку порядок загрузки классов-компонентов JavaBeans управляется пользователем. Стратегия подменять "на лету" один такой компонент на другой (после чего удалять первый) чревата ошибками. Анализировать же всякий класс как компонент JavaBeans нерационально с точки зрения производительности и попросту некорректно.

Для преодоления отмеченных трудностей был реализован гибридный компонент-обертка, осуществляющий "ленивую" интроспекцию классов Java. Иными словами, такой компонент ведет себя как "непрозрачный" (и не пытается анализировать структуру инкапсулируемого класса) до тех пор, пока это не потребуется явно.

Второй сложностью интеграции с JavaBeans является то, что компонентная модель данной технологии включает такие понятия, как методы и события (отличные от PropertyChangeEvent). И те, и другие представляются в виде свойств предложенной модели. При этом соответствующие методам свойства доступны для записи (и, как следствие, связывания), а событиям — только для связывания.

[7] Данный подход перекликается с такими актуальными практиками, как разработка, основанная на тестировании (TDD), и проектирование на основе примеров (design by example).

Результаты и направления дальнейших исследований

Располагая поддержкой RTTD и возможностью использовать компоненты JavaBeans в качестве скомпилированных компонентов, удалось воспользоваться существующей реализацией стандартных типов узлов VRML в виде библиотеки компонентов JavaBeans [1] (рис. 3), чтобы реализовать модель VRML (в том числе, и механизм PROTO).

* * *

Возможность динамически создавать новые компоненты на основании существующих способна расширить сферу применения технологий компонентной разработки ПО. Впрочем, как и в случае любого новшества, вносимого в инструментарий разработчиков, потребуется время, прежде чем будут найдены оптимальные способы их использования.

В данной научной работе использованы результаты проекта "Компонентная модель для организации взаимодействия физической и виртуальной реальности", выполненного в рамках Программы фундаментальных исследований НИУ ВШЭ в 2012 г.

Список литературы

1. **Гринкруг Е. М.** Использование JavaBeans-компонент в 3D-моделировании // Бизнес-информатика. 2010. № 3 (13). С. 47—56.

2. **Фаулер М.** Предметно-ориентированные языки программирования. М.: Вильямс, 2011.

3. **Aldrich J.** Using Types to Enforce Architectural Structure // Proceedings of the Working International Conference on Software Architecture (WICSA '08). 2008. URL: http://www.cs.cmu.edu/~aldrich/papers/wicsa08.pdf

4. **Bruneton E., Coupaye T., Stefani J.B.** The Fractal Component Model specification. Version 2.0-3. The ObjectWeb Consortium, 2004. URL: http://fractal.ow2.org/specification/index.html

5. **International** Standard ISO/IEC 14772-1:1997. The Virtual Reality Modeling Language. URL: http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/

6. **International** Standard ISO/IEC 19775-1.2:2008. X3D Architecture and base components Edition 2. URL: http://www.web3d.org/files/specifications/19775-1/V3.2/index.html

7. **McIlroy M.D.** Mass produced software components // Software Engineering, Report on a conference sponsored by the NATO Science Committee. 7—11 October 1968. Garmisch, Germany / ed. P. Naur, B. Randell. Brussels: Scientific Affairs Division, NATO, 1969. P. 138-155.

8. **Object** Management Group. The Common Object Request Broker: Architecture and Specification. Version 3.1. Part 3 — Components. OMG document formal/2008-01-08, 2008. URL: http://www.omg.org/spec/CORBA/3.1/Components/PDF

9. **Redmond F. E.** DCOM: Microsoft Distributed Component Object Model. Foster City: IDG Books Worldwide Inc., 1997.

10. **Sun** Microsystems Inc. The JavaBeans API specification. Version 1.01-A, Sun Microsystems Inc., 1997. URL: http://download.oracle.com/otn-pub/jcp/7224-javabeans-1.01-fr-spec-oth-JSpec/beans.101.pdf

11. **Szyperski C.** Component Software: Beyond Object-Oriented Programming. 2nd ed. Boston: Addison-Wesley Professional, 2002.

ПАМЯТИ КОЛЛЕГИ И ТОВАРИЩА

НОРЕНКОВ
Игорь Петрович
(19.08.1933—31.12.2012)

На 80-м году ушел из жизни известный ученый, блестящий преподаватель, талантливый организатор и замечательный человек, член редколлегии журнала "Программная инженерия" — Игорь Петрович Норенков.

Основатель и до последнего времени руководитель кафедры систем автоматизированного проектирования МГТУ им. Н. Э. Баумана, Заслуженный деятель науки и техники РФ, лауреат Государственной премии СССР, доктор технических наук, один из ведущих специалистов в области современных информационных технологий, Игорь Петрович внес неоценимый вклад в развитие российской науки и образования.

И. П. Норенков родился 19 августа 1933 г., в 1960 г. закончил МВТУ им. Н. Э. Баумана (ныне МГТУ им. Н. Э. Баумана) по специальности "Математические машины". В 1975 г. за большой вклад в теорию и практику методов автоматизированного проектирования электронных схем Игорь Петрович был удостоен Государственной премии СССР.

Игорь Петрович являлся одним из пионеров компьютеризации проектно-конструкторской деятельности, руководителем одной из ведущих отечественных школ в области моделирования и оптимизации. И. П. Норенков в 1982 г. основал в МВТУ им. Н. Э. Баумана одну из первых отечественных кафедр в области систем автоматизированного проектирования, которая занимает ведущее место в системе подготовки высококвалифицированных специалистов в области автоматизации проектирования.

Все знали И. П. Норенкова как высокоинтеллегентного ученого, которого отличали широкий научный кругозор, постоянное стремление двигаться вперед, доброжелательность, отзывчивость. В памяти друзей и коллег он останется очень светлым человеком.

Редколлегия и редакция журнала "Программная инженерия"
выражают глубокое соболезнование родным и близким покойного.
Вечная память о Игоре Петровиче Норенкове навсегда сохранится в наших сердцах.

УДК 004.4+004.7

Ю. А. Богоявленский, канд. техн. наук, зав. каф., Петрозаводский государственный университет,
e-mail: ybgv@cs.karelia.ru

Прототип экспериментальной платформы Nest для исследования моделей и методов управления ИКТ-инфраструктурами локальных поставщиков услуг Интернет

Предложена архитектура и представлены подсистемы экспериментальной платформы Nest, опирающейся на модель архитектуры предприятия — локального поставщика услуг Интернет, объектный граф которой хранится в базе данных. Это позволяет обеспечить доступ к данным различных измерений при рабочих нагрузках, близких к реальным, структурируя эти данные по аппаратным, организационным и пространственным единицам и их произвольным агрегациям.

Ключевые слова: *модели сетевого управления, поставщики услуг Интернет, экспериментальные платформы, архитектура предприятия, объектные модели, обнаружение топологии, визуализация сетей, данные измерений*

Введение

ИКТ-инфраструктуры (далее Сетей) предприятий — поставщиков сетевых услуг (ПСУ) являются сложными системами, методы управления которыми требуют существенного развития. В отчете [1] Винтон Серф с соавторами пишут: "В настоящее время сетевое управление характеризуется недостатком информации о статусе и работоспособности сетей, лавиной данных (объемных, неоднозначных, неполных и противоречивых одновременно) и грубыми или неточными управляющими механизмами, чье воздействие трудно предсказуемо... Для изменения сегодняшнего статус-кво необходимы глубокие исследования в области сетевого управления". Авторы работы [2] отмечают, что дорогостоящие продукты, созданные индустрией для достижения должного уровня мониторинга и управляемости Сетей, не решают этих задач, а научные методы анализа систем сетевого управления находятся в примитивном состоянии. Разработка формальных методов анализа этих систем должна опираться на данные измерений и сравнительные тесты, проводимые при рабочих нагрузках и процессах, близких к реальным, что, в свою очередь, требует разработки экспериментальных платформ (ЭП, английский термин *testbed* [3, 4]) путем встраивания их в существующие Сети (*overlay testbeds* [3]).

Разрабатывая ЭП разумно использовать идеи существующих функциональных и концептуальных моделей FCAPS [5], TMN [6] (появляется уровень бизнес-процессов) и Frameworx [7]. Последняя предоставляет комплексную модель архитектуры ИКТ-инфраструктуры, бизнес-процессов и принадлежит активно развиваемому направлению интегрированного моделирования архитектуры предприятия в целом (ИМАП — IEM — *Integrated Enterprise Modelling* [8, 9]), когда Сеть рассматривается как подсистема. В рамках ИМАП разработаны различные инструментальные средства и стандарты [10—12], в том числе содержащие сотни классов, интегрированные объектные модели CIM [13] и SID [14].

Распределение числа ПСУ по уровням

Номер и название уровня	Число ПСУ
0. Плотное ядро	20
1. Транзитное ядро	129
2. Внешнее ядро	897
3. Малые региональные ПСУ	971
4. Локальные ПСУ	8898

Представляется, что с помощью ЭП целесообразно обосновывать формальные методы решения не только классических задач сетевого управления, но и задач исследования эффективности использования Сети в бизнес-процессах. Естественно, что эти два класса задач тесно связаны. Например, в задачах анализа производительности и планирования мощности модели FCAPS необходимы характеристики или модели рабочей нагрузки Сети, которые удобно строить на основе ИМАП-моделей.

Важной характеристикой ЭП является уровень ПСУ, подлежащих исследованию. В зависимости от этого уровня (*tier*) ПСУ выполняют существенно различные функции в Интернет. В таблице [15] приведено распределение числа ПСУ по уровням в Интернете (начиная с уровня 0 — магистральные ПСУ или плотное ядро, англ. *backbone*).

Большинство существующих ЭП (PlanetLab, Geni, другие [3]) предназначены для исследования новых механизмов и сервисов Интернет в интересах ПСУ уровней 0 и 1. В то же время не удалось найти в литературе описаний разработок ЭП для исследования самого многочисленного класса — локальных ПСУ (лПСУ), которые находятся на 3 и 4 уровнях.

Предприятия лПСУ относятся к классу объектов средней величины, их Сети могут содержать до нескольких тысяч оконечных ЭВМ и небольшое число пограничных маршрутизаторов. Эти предприятия ближе всех к пользователям, их основная функция — поставка услуг своему персоналу, хотя иногда они выполняют небольшой объем услуг по транзиту трафика. Следует отметить, что масштабы разработки методов управления и проектирования, ориентированных на Сети лПСУ (англ. *Enterprise Networks*) в последние годы значительно расширились [16]. Однако на настоящее время они исследованы гораздо слабее, чем глобальные сети, а сложности разработки этих методов для лПСУ не меньше, а иногда и больше, чем для последних [17].

Таким образом, актуальной является задача разработки и реализации ЭП для исследования моделей и методов управления Сетями лПСУ и эффективности использования этих Сетей в бизнес-процессах. ЭП Nest [18], рассматриваемая в настоящей статье, ориентирована на решение именно этой задачи.

Архитектура ЭП Nest

При разработке Nest были приняты следующие принципиальные архитектурные решения.

- Основная задача ЭП — обеспечение доступа к данным измерений, структурированного по элементарным пространственным, организационным и аппаратно-программным единицам архитектуры предприятия и их произвольным агрегациям. Сложность моделей архитектуры, подобных CIM [13], SID [14], затрудняет их использование для решения этой задачи. В Nest используется описываемая ниже разработанная под руководством автора простая объектная модель SON архитектуры лПСУ.
- Объектный граф модели SON конкретного лПСУ представляет собой граф архитектуры лПСУ. Он хранится в объектно-ориентированной базе данных (ООБД) под управлением подсистемы NestDB. Необходимые исследователю файлы с данными измерений хранятся в файловой системе в собственных форматах. Структурирование этих данных согласно модели SON выполняется путем передачи результатов запроса структурной информации к NestDB в подсистему унифицированного доступа к данным NestData.
- Информация об аппаратных элементах Сети и их связях (далее граф Сети) является важным инструментом в системах сетевого управления и ЭП. Так, задачу планирования мощности можно решать с помощью имитационных моделей на базе этого графа [19]. Графы Сети современных лПСУ содержат тысячи элементов и связей, существенно меняются во времени, поэтому для их использования необходима система автоматизированного построения и хранения в БД. В Nest входит подсистема NesTopo, поддерживающая в автоматизированном режиме актуальное состояние графа Сети.
- Визуальные образы могут кардинально увеличивать скорость восприятия данных [20]. Область визуализации топологии сетей разного масштаба интенсивно развивается. Авторы работы [21] отмечают пользу инструментальных средств визуализации для администраторов, исследователей и пользователей сетевых систем. В этой же работе представлена развернутая классификация более 120 средств визуализации по масштабу сетевой системы, степени детализации, методам визуального отображения графа и другими признаками (всего по 11). Администраторы используют этот инструментарий для проектирования и обоснования модернизации инфраструктуры, сопровождения, мониторинга, изоляции ошибок, анализа производительности, показа диагностики связности [22] и даже решения задач безопасности [23].

Риски возникновения брешей в системах безопасности возрастают как в связи с увеличением числа удаленных сотрудников и малых подразделений, так и с упрощением доступа к сетевым технологиям, позволяющим обходить сетевые экраны. Сейчас, например, можно очень легко приобрести и подключить к сети несанкционированное устройство, например кабельный модем, беспроводную точку доступа или мощный смартфон.

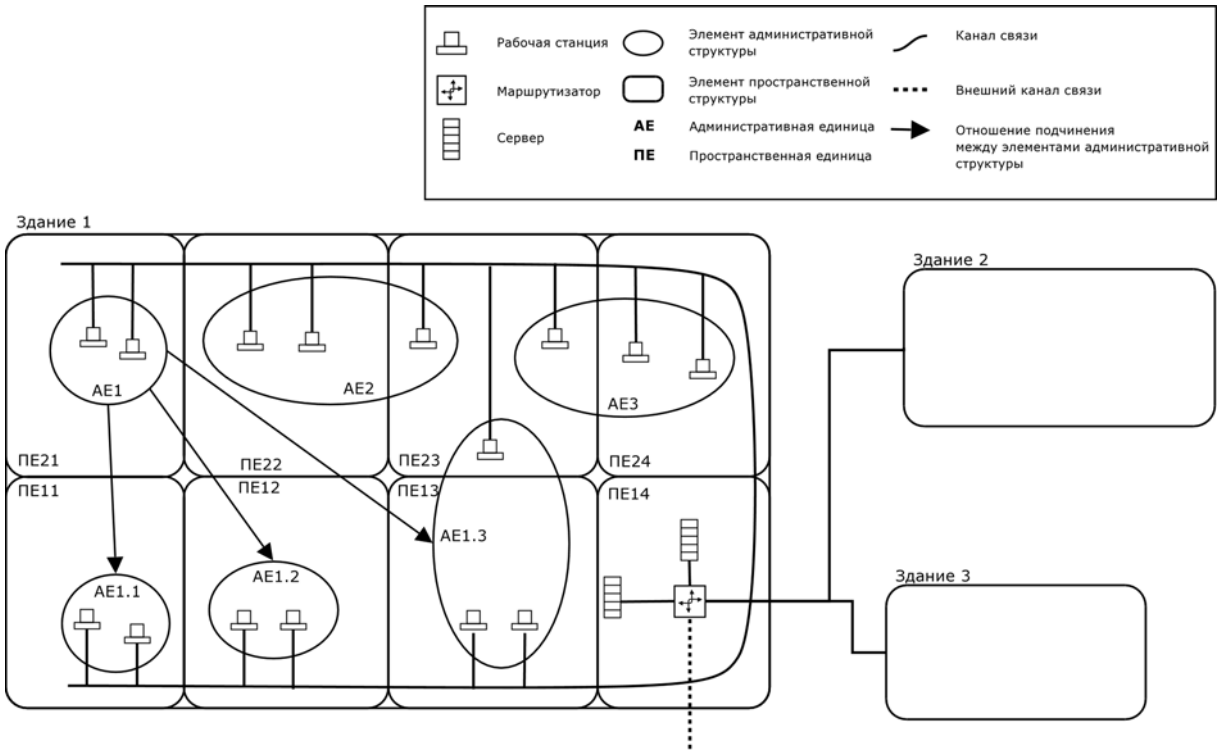


Рис. 1. Схема архитектуры лПСУ

Исследователи могут строить топологии на базе теоретических положений, генерировать топологии для различных задач, изучать поведение новых алгоритмов и протоколов, проверять адекватность моделей. При этом значительную помощь могут оказать визуальные модели, дополненные необходимыми данными.

В Nest входит подсистема NestViz, обеспечивающая гибкое управление формированием интерактивных визуальных образов подграфов графа архитектуры лПСУ с помощью разработанного предметно-ориентированного языка.

- Исследователю могут понадобиться данные измерений, представляемые в разных форматах (потоки трафика, регистрационные файлы и т. п.). Подсистема NestData обеспечивает унифицированный доступ к этим данным.
- В качестве основного инструментария для наблюдения за трафиком Сети используется развивающаяся и стандартизованная организацией IETF измерительная технология потоков данных [24, 25], которая широко используется для решения различных задач сетевого управления (см. ссылки в работе [26]).

Базовая модель архитектуры локального поставщика услуг Интернет

Модель SON соответствует представленной на рис. 1 схеме архитектуры лПСУ. Сеть располагается в нескольких зданиях, при этом ее элементы обслуживают различные организационные единицы.

SON — это композиция трех моделей основных структур лПСУ, а именно пространственной (*S — Spatial*), организационной (*O — Organizational*) и аппаратно-сетевой (*N — Network*). Модель представлена на рис. 2 в нотации UML.

Модель пространственной структуры не требует комментариев. В моделе организационной структуры абстрактный класс AbstractOU представляет обобщен-

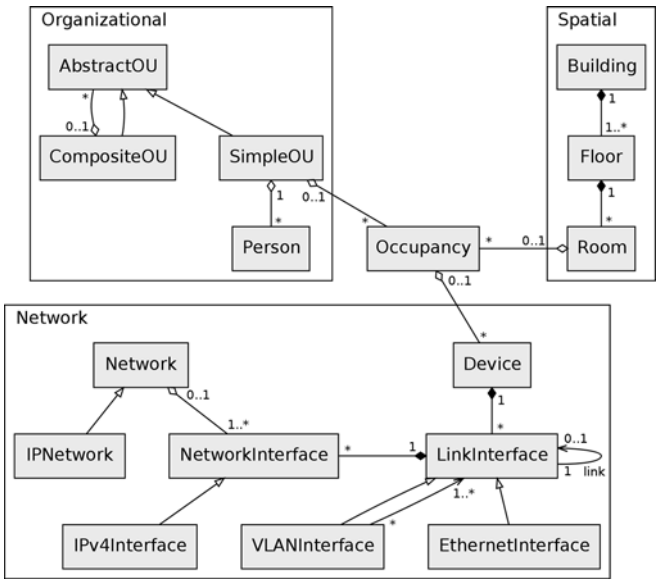


Рис. 2. Модель SON

ную организационную единицу, а его наследники — простейшие (SimpleOU) и составные (CompositeOU) единицы.

Сетевые устройства представлены классом Device, их канальные интерфейсы — классом LinkInterface. Его наследники представляют интерфейсы конкретных канальных протоколов (EthernetInterface), виртуальных сетей стандарта IEEE 802.1Q (VLANInterface) и аналогичных. Объекты класса VLANInterface определяют принадлежность групп канальных интерфейсов к виртуальным сетям. Сетевые интерфейсы (если они есть) объектов класса LinkInterface представлены классом NetworkInterface (например, IPv4Interface). Его объекты могут объединяться в сети (Network) — произвольные группы интерфейсов сетевого уровня, в частности, в подсети IP (IPNetwork).

Модели *S*, *O* и *N* структур связаны следующими отношениями. Комната может содержать сетевые ус-

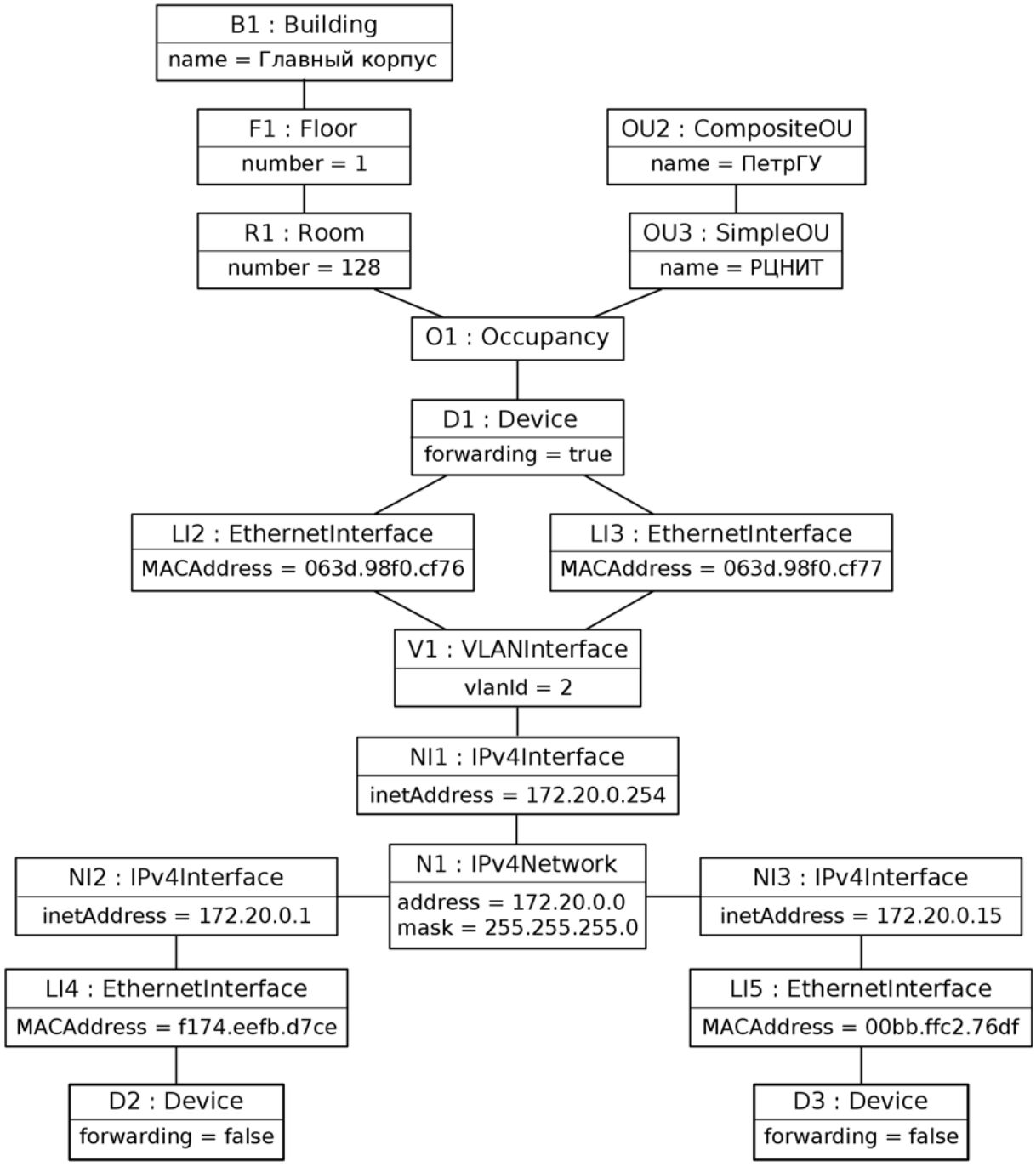


Рис. 3. Пример UML-диаграммы объектов графа архитектуры лПСУ

тройства. Организационная единица может иметь несколько комнат и устройств. Комната может содержать устройства, принадлежащие разным организационным единицам, что требует связей "многие ко многим" между классами Room и SimpleOU.

Класс Occupancy имеет связи "многие к одному" с классами Room и SimpleOU, а также связь "один ко многим" с классом Device, что устраняет необходимость связи "многие ко многим" между классами Room и SimpleOU. Этот класс представляет часть комнаты, занимаемую организационной единицей, и определяет пару Room—SimpleOU, однозначно задающую расположение устройств в пространственной и организационной структурах.

В конкретный момент времени экземпляры классов являются вершинами объектного графа, представляющего архитектуру лПСУ. Дуги графа считаются ненаправленными, строятся согласно связям между классами и интерпретируются по определенному в модели смыслу, как это указано в примере на рис. 3. В главном корпусе B1 на первом этаже F1 в кабинете 128 R1 находится маршрутизатор D1, имеющий канальные интерфейсы LI2, LI3. Они объединены в виртуальную сеть V1, имеющую сетевой интерфейс NI1, принадлежащий подсети N2, содержащей также сетевые интерфейсы NI2, NI3 устройств D2, D3. Маршрутизатор принадлежит организационной единице OU3 РЦНИТ (Региональный центр новых информационных технологий), входящей в OU2 (Петрозаводский государственный университет).

Содержащая всего 16 классов модель SON намного проще большинства существующих моделей, однако она позволяет выполнять полноценный структурированный доступ к данным измерений, соответствующий архитектуре лПСУ. Элементы компонентов SON и их агрегации (например, этажи здания, отделы организации, IP подсети, виртуальные сети) могут трактоваться как объекты измерений, в том числе связанных с бизнес-процессами организационных единиц. SON также может быть связана с более сложными моделями архитектуры организации.

Объектная база данных Nest

Текущий объектный граф архитектуры лПСУ хранится в ООБД. При этом граф Сети поддерживается автоматизировано. Операции БД в объектных терминах выполняются с помощью программного каркаса Hibernate, реализующего метод объектно-реляционного отображения Java Persistence API (JPA). Реляционная схема, соответствующая модели SON, генерируется автоматически. Запросы формулируются на языке HQL и также автоматически транслируются в SQL-запросы к базовой реляционной БД (в Nest это MySQL).

Результатом запроса являются значения атрибутов нужных элементов архитектуры лПСУ или их агрегаций. Эти результаты могут далее обрабатываться другими подсистемами Nest или программами исследователя.

Далее представлены примеры запросов на HQL.

Какой организационной единице принадлежит устройство с IP адресом 192.168.112.48:

```
select sou from SimpleOU soujoin sou.occupancies os
join os.devices d join d.linkInterfaces li join li.networkInterfaces ni where ni.inetAddress='192.168.112.48'
```

Результат: ([Кафедра ИМО]).

Вывести IP- и MAC-адреса ЭВМ, принадлежащих кафедре ИМО:

```
select ipv4.inetAddress, ei.MACAddress from IPv4Interface ipv4
join ipv4.linkInterface ei
where ipv4.linkInterface.device.occupancy.OU.name=' Кафедра ИМО'
```

Результат:

```
192.168.112.40 00:14:85:04:37:39
192.168.112.28 00:1E:58:2B:C1:D7
192.168.112.57 00:13:46:77:33:B8
...
192.168.112.56 00:21:91:F4:A1:02.
```

Запросы и их результаты представляются также через интерфейс Nest. На рис. 4 показаны результаты запроса сетевой пространственной и организационной

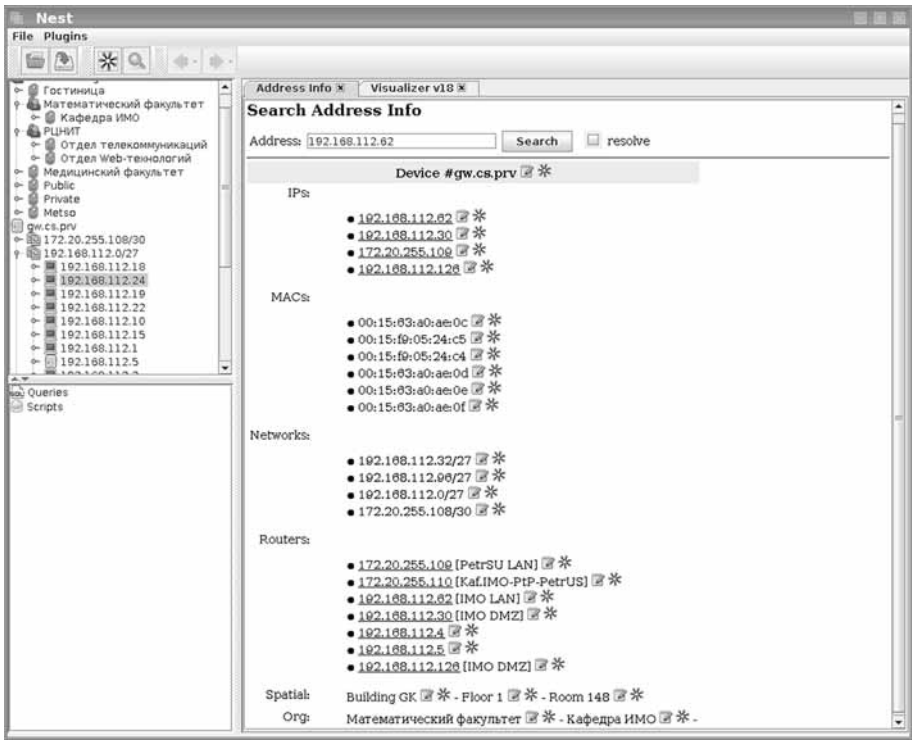


Рис. 4. Результат запроса о выводе данных об устройстве по его IP-адресу

информации о маршрутизаторе кафедры информатики и математического обеспечения. Нажатие на значок справа от данных обеспечивает переход к редактированию (пиктограмма блокнота) или визуализации (пиктограмма графа) элемента.

Автоматизированная поддержка графа Сети

В ЭП Nest граф Сети строится подсистемой NesTopo, которая, в отличие от предыдущих разработок, не только обнаруживает аппаратные элементы сети и связи между ними, но и размещает в БД данные о графе, соответствующие модели SON, поддерживая их актуальность.

Физической топологии в зависимости от уровня абстракции могут соответствовать различные логические топологии. В текущей версии топология определяется на уровне IP, что достаточно для исследования моделей и методов управления сетями на уровне потоков данных. Для предполагаемой в будущем реализации алгоритма построения топологии канального уровня на основе протокола LLDP [27] в БД сохраняются данные о физических интерфейсах.

Экземпляры классов модели, соответствующие узлам актуального графа Сети, определяются путем обхода маршрутизаторов с опросом данных их MIB по протоколу SNMP. Установка дополнительного ПО на сетевые устройства не требуется. Состав IP-подсетей определяется с точностью до периода обновления ARP-кешей. Более подробное описание см. в работе [28].

Экспериментальная эксплуатация подсистемы NesTopo показала ее хорошую производительность. Формирование БД графа Сети ПетрГУ, содержащей примерно 4500 объектов, требует около 60 с при объеме исходящего трафика около 60 Кбайт, входящего — 350 Кб. В настоящее время разработана и готовится к публикации значительно ускоряющая построение графа параллельная версия алгоритма, реализованная средствами пакета java.util.concurrent из состава инструментального пакета JDK7.

Визуализация графа архитектуры лПСУ

Подсистема визуализации должна поддерживать перемещение, масштабирование, интерактивное изменение параметров раскладки, произвольное агрегирование/деагрегирование и сокрытие/раскрытие фрагментов.

Разнообразие задач по разработке моделей и методов сетевого управления определяет необходимость предоставления пользователю гибкого инструментария для построения самых различных интерактивных визуальных образов подграфов графа архитектуры лПСУ как на уровне элементарных единиц, так и на уровне их произвольных агрегаций.

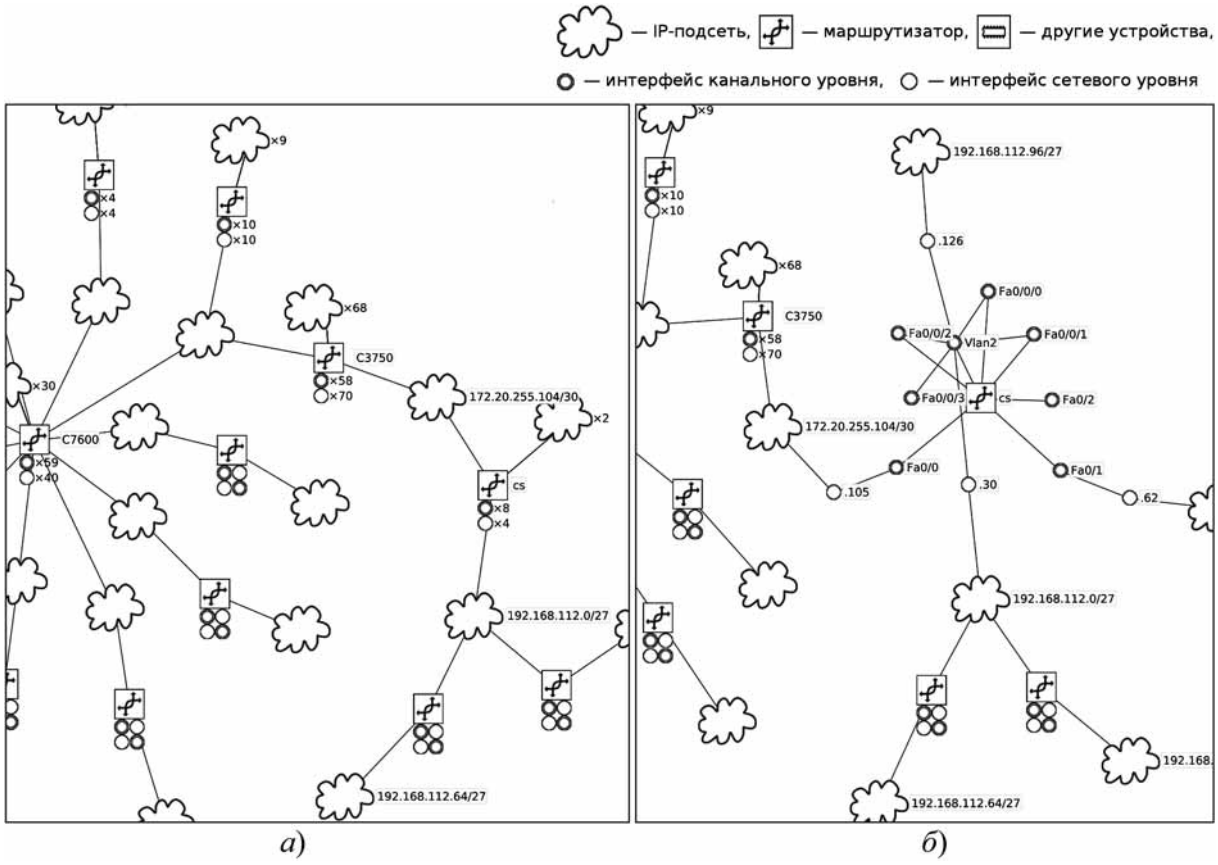


Рис. 5. Фрагмент визуального образа Сети ПетрГУ

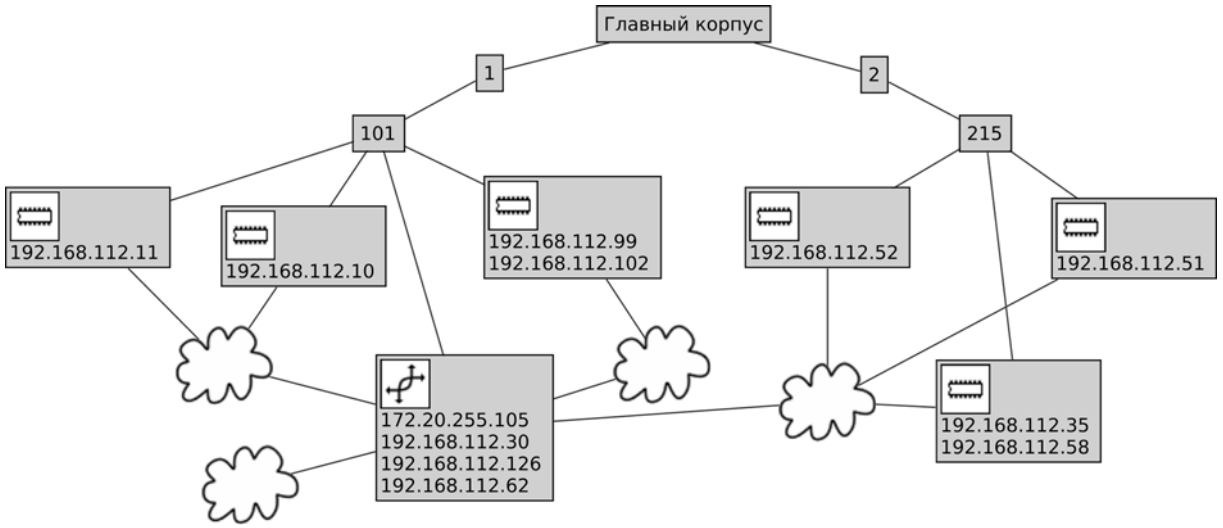


Рис. 6. Пространственное расположение устройств и сетей

```
;; Здания и этажи.
[Building Floor] #(-> % stack (lesser-y 0))

;; Кабинеты 101 и 215, Occupancy скрываются.
(path [{:number "101"} {:number "215"}] Occupancy)
 #(-> % first combined (const-y 60))

;; Устройства вместе с их интерфейсами.
(path Device LinkInterface NetworkInterface)
 (fn [[device & ifs]]
 (-> ifs
 ;; Список сетевых интерфейсов
 (->> (filter (partial instance? NetworkInterface))
 ;; Для каждого сетевого интерфейса создать текстовую
 ;; метку с описанием.
 (map (comp label display-string))
 ;; Расположить вертикально вместе с иконкой устройства;
 ;; combined — иконка + свойства при наведении курсора.
 (apply vbox (combined device)))
 ;; Добавить фон и отступ.
 (panel 2)
 ;; Добавить рамку.
 border
 ;; Ограничить координату y.
 (greater-y 120)))

;; Сети
Network stack
```

Рис. 7. Правила для формирования образа, представленного на рис. 6

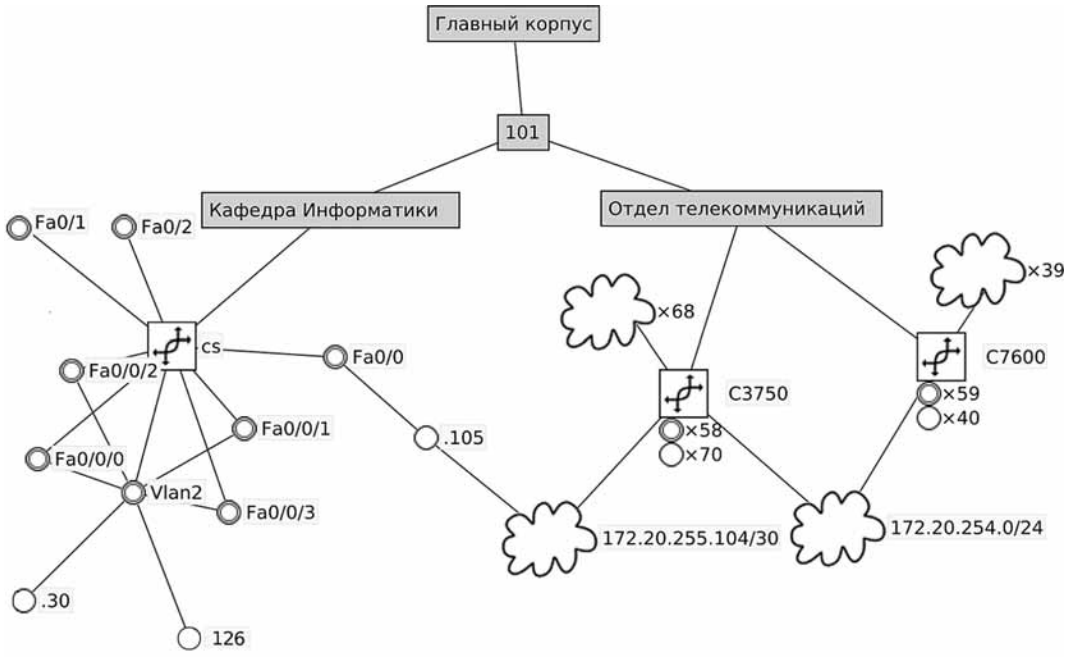


Рис. 8. Принадлежность устройств из кабинета 101 к организационным единицам

В коммерческих системах, например IBM Tivoli [29] и HP Network Node Manager, набор визуальных образов фиксирован. Универсальные системы, такие как Graphviz [30] и Gephi [31], предоставляют больший выбор алгоритмов раскладки графов, а Gephi поддерживает интерактивность. Однако они не имеют гибких пользовательских средств формирования графа и его образа на основе данных объектного графа.

В качестве такого средства разработан предметно-ориентированный язык на базе языка Clojure [32] (современная версия Lisp, выполняющаяся на JVM, нацеленная на параллельные вычисления и имеющая развитую макросистему). При визуализации из графа архитектуры лПСУ по правилам языка выбирается и преобразуется в граф интерактивных визуальных объектов требуемый подграф. Из него с помощью алгоритма раскладки графа (и, возможно, с применением правил языка) формируется визуальный интерактивный образ.

Раскладка выполняется по методу физических аналогий, учитываются размеры вершин. При этом пересчеты сил для каждой вершины визуального отображения, определенной правилами видимой части графа, и миниатюрного образа всего графа выполняются параллельно.

Для работы с интерактивными визуальными объектами разработана специальная библиотека, базовые элементы интерфейса которой не имеют состояния. Она позволяет строить динамические интерфейсы с большим числом интерактивных элементов и обеспечивает многопоточное выполнение. Рассмотрим примеры построенных визуальных объектов. На рис 5. представлены фрагменты визуального образа Сети ПетрГУ. На рис. 5, *а* отображены только маршрутизаторы и подсети. Под маршрутизаторами показано число интерфейсов канального уровня (пара концентрических окружностей) и сетевого уровня (простая окружность).

Система визуализации позволяет интерактивно переопределить правила визуализации для подграфа. На рис. 5, *б* изображен фрагмент, в котором для отображения маршрутизатора cs теперь используется другой набор правил визуализации: каждый элемент объектного графа отображается в виде отдельной вершины — иконки с подписью. При этом для маршрутизатора cs просматривается соответствие между его интерфейсами Fast Ethernet (Fa), виртуальным интерфейсом VLAN, сетевыми интерфейсами IPv4 и IP-подсетями.

На рис. 6 представлен визуальный образ, иллюстрирующий расположение в пространстве некоторых сетевых устройств кафедры Информатики и математического обеспечения (кабинет 101 на 1-м этаже, кабинет 215 на 2-м этаже). Образ получен применением правил языка, представленных на рис. 7.

Образ на рис. 8 иллюстрирует принадлежность находящихся в кабинете 101 (серверная) сетевых устройств к организационным единицам ПетрГУ.

Структура и функции подсистемы NestViz описаны в работе [33].

Унифицированный доступ к данным измерений

Исследователю могут понадобиться данные измерений из различных источников (регистрационные файлы, системы мониторинга, коллекторы записей о потоках данных). Подсистема NestData обеспечивает унифицированный доступ к данным, предоставляя абстракции источника данных (класс RecordSource) и минимальной единицы данных (класс Record), а также расширяемые механизмы преобразования данных в формат абстракций.

Особенности доступа к конкретным источникам данных, интерфейс настройки, разбор и генерация соответствующих объектов-записей реализуются в наследниках класса RecordSource. Записи, получаемые от конкретного источника, описываются классом, наследующим от Record. Существует реестр реализаций абстракций для различных источников данных.

В настоящее время реализован класс FlowRecord, имеющий свойства, соответствующие атрибутам записей о потоках данных, накопленных коллекторами NetFlow в формате пакета flow-tools [34]. Преобразование записей во внутреннее представление Nest выполняет библиотека FlowBrook. Подсистема NestData обеспечивает выборку потоков для заданного временного промежутка, фильтрацию и группировку потоков по значениям их атрибутов.

Рассмотрим схему работы NestData на примере задачи определения объема трафика, переданного между двумя организационными единицами на заданном временном интервале. На первом шаге выполняются два запроса к NestDB, выдающие множества IP-адресов устройств, связанных с каждой из организационных единиц. На втором шаге эти множества трансформируются в фильтр для отбора нужных потоков. На третьем шаге NestData, используя временной интервал и фильтр, получает соответствующий набор записей и вычисляет искомый объем трафика.

Реализация платформы Nest

Nest разрабатывается на платформе OpenJDK 6 (языки Java и Clojure).

Исследовательский характер Nest определяет необходимость разработки простого механизма, минимизирующего изменения в программном коде, необходимые при модификации объектной модели SON. Этот механизм реализуется макросредствами Clojure. Разработанный с этой целью простой набор макроопределений позволяет описывать на высоком уровне классы и связи модели и генерировать код Clojure, содержащий необходимые для реализации классов функции и структуры данных. При генерации в код также включаются фрагменты, поддерживающие целостность связей в объектном графе и отслеживающие его динамические изменения. Транслируя этот код, компилятор Clojure получает байт-код классов Java.

На Clojure реализована также подсистема NestViz и механизмы фильтрации потоков NetFlow. Остальные компоненты выполнены на языке Java. Текущая версия Nest содержит (без комментариев и пустых строк) 18512 строк на языке Java и 7321 на языке Clojure.

Заключение

По оценкам ведущих исследователей Интернет, научные методы анализа систем сетевого управления находятся в примитивном состоянии, а разработка таких методов должна опираться на данные измерений и сравнительные тесты, проводимые на ЭП в условиях реалистичных рабочих нагрузок и процессов. Разработки таких платформ для наиболее близких к потребителям локальных поставщиков услуг практически отсутствуют.

В статье предложена архитектура и представлены подсистемы прототипа экспериментальной платформы Nest, опирающиеся на модель предприятия — локального поставщика услуг Интернет, объектный граф которой хранится в базе данных. Это позволяет обеспечить доступ к данным различных измерений при рабочих нагрузках, близких к реальным, структурируя эти данные по аппаратным, организационным и пространственным единицам и их произвольным агрегациям.

Прототип реализован в среде Java. Актуальное состояние графа сетевой структуры на уровне IP поддерживается в базе данных в автоматизированном режиме. Подсистема визуализации позволяет гибко задавать свойства визуальных образов с помощью предметно-ориентированного языка. Подсистема унифицированного доступа к данным предоставляет абстрактные инструментальные средства описания источников данных и механизмов доступа к ним. В настоящее время реализован доступ к данным, получаемым по технологии NetFlow.

В будущем планируется расширять набор источников данных и перевести прототип Nest в режим опытной эксплуатации.

Автор благодарен А. В. Воронину и Н. С. Рузановой за внимание и поддержку данной работы, М. А. Крышеню, А. С. Колосову, В. М. Димитрову и К. А. Кулакову за полезные идеи, обсуждения и упорную работу по реализации системы, В. А. Пономареву, И. О. Суворову и И. А. Зиновику за помощь в выполнении экспериментов.

Список литературы

1. **Cerf V., Davie B., Greenberg A.** et al. FIND Observer Panel Report: Tech. rep.: NSF NeTS FIND Initiative, 2009. April. URL: http:// www.nets-find.net/FIND_report_final.pdf.

2. **Al-Shaer E., Greenberg A., Kalmanek C.** et al. New Frontiers in Internet Network Management // Computer Communication Review. 2009. Vol. 39. P. 37—39.

3. **Report** of NSF Workshop on Network Research Testbeds. 2002. November. URL: http://www-net.cs.umass.edu/testbed_workshop/testbed_ workshop_report_final.pdf.

4. **GENI.** Exploring Network of the Future. URL: http://www.geni.net/

5. **Basic** Reference Model, Part 4, Management Framework. ISO/IEC 7498- 4:1989. Geneva, Switzerland. ISO, 1989.
6. **Principles** for a Telecommunications management network. ITU-T Recommendation M.3010. International Telecommunication Union, 1996.
7. **TM Forum** — Frameworx. URL: http://www.tmforum.org/Frameworx/1911/home.html.
8. **Mertins K., Jochem R.** Integrated Enterprise Modeling: Method and Tool // SIGGROUP Bulletin. 1997. Vol. 18. P. 63—66.
9. **Данилин А. В.** Архитектура предприятия, учебный курс для руководителей информационных служб. Интернет-университет информационных технологий. URL: http://www.intuit.ru/department/itmngt/entarc/.
10. **Leist S., Zellner G.** Evaluation of Current Architecture Frameworks // Proceedings of SAC'06. Dijon, France, 2006. P. 1546—1553.
11. **Enterprise integration** — Framework for enterprise modelling. ISO 19439:2006. Geneva, Switzerland. ISO, 2006.
12. **The Casewise** Product Portfolio. URL: http://www.casewise.com/Products/.
13. **Distributed** Management Task Force. Common Information Model (CIM) Infrastructure. 2009. URL: http://dmtf.org/sites/default/files/standards/documents/DSP0004_2.5.0.pdf
14. **TM Forum** — Information Framework (SID). URL: http://www.tmforum.org/BestPracticesStandards/InformationFramework/6647/Home.html.
15. **Subramanian L., Agarwal S., Rexford J., Katz R. H.** Characterizing the Internet Hierarchy from Multiple Vantage Points: Tech. Rep. UCB/CSD-1-1151. California: Computer Science Division (EECS) University of California Berkeley, 2001.
16. **Proceedings** of USENIX INM/WREN'10. 2010. URL: http://www. usenix.org/event/inmwren10/tech/
17. **Sung Y.-W. E., Rao S. G., Xie G. G., Maltz D. A.** Towards Systematic Design of Enterprise Networks: Tech. Rep. Paper 375. Purdue University, ECE, 2008. URL: http://docs.lib.purdue.edu/ecetr/375/.
18. **Богоявленский Ю. А.** Nest: экспериментальная платформа для исследования моделей и методов управления сетями локальных поставщиков услуг интернет // Труды XIX Всероссийской научно-методической конференции Телематика'2012. Том 2. С. 251—253.
19. **Claise B., Wolter R.** Network Management: Accounting and Performance Strategies. Cisco Press, 2007. 631 p.
20. **Захарова А. А., Шкляр А. В.** Визуальные модели // Проблемы информатики. 2012. № 1. С. 41—46.
21. **Di Battista G., Rimondini M.** Computer Networks Roma Tre University,. 2010. URL: http://www.cs.brown.edu/~rt/gdhandbook/chapters/networks.pdf
22. **Morgan D.** Network Topology: Connectivity Visualized, Microsoft Corporation, 2005. URL: http://download.microsoft.com/download/9/8/f/98f3fe47-dfc3-4e74-92a3-088782200fe7/TWMO05006_WinHEC05.pp
23. **Wang W., Lu A.** Visualization Assisted Detection of Sybil Attacks in Wireless Networks // Proceedings of the VizSEC '06. ACM Inc., 2006. P. 51—60.
24. **Claise B.** Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101. IETF, 2008.
25. **Cisco** IOS NetFlow — Cisco Systems. URL: http://www.cisco.com/go/netflow.
26. **Sekar V., Reiter M. K., Zhang H.** Revisiting the Case for a Minimalist Approach for Network Flow Monitoring // Proc. of IMC'10. Melbourne, Australia. November. 2010. P. 328—341.
27. **IEEE** Standard for Local and metropolian area networks — Station and Media Access Control Connectivity Discovery. IEEE Std 802.1AB-2009.
28. **Богоявленский Ю. А., Колосов А. С.** Организация и автоматизированная поддержка объектной базы данных графа ИКТ-инфраструктуры поставщика услуг Интернет // Научно-технические ведомости СПбГПУ. Серия "Информатика. Телекоммуникации. Управление". 2011. № 3 (126). С. 27—36.
29. **IBM** Tivoli Network Manager Version 3.7 Topology Visualization Guide. 2007. URL: http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.itnetmantip.doc_3.7/7962/NMIP_visualization.pdf.
30. **Ellson J., Gansner E. R., Koutsofios E.** et al. Graphviz and dynagraph — static and dynamic graph drawing tools // Graph drawing software. Springer-Verlag, 2003. P. 127—148.
31. **Bastian M., Heymann S., Jacomy M.** Gephi: An Open Source Software for Exploring and Manipulating Networks. 2009. URL: http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154.
32. **Clojure — home.** URL: http://clojure.org/
33. **Крышень М. А., Богоявленский Ю. А.** Интерактивная визуализация графа ИКТ-инфраструктуры предприятия // Научно-технические ведомости СПбГПУ. Серия "Информатика. Телекоммуникации. Управление". 2011. № 3 (126). С. 172—175.
34. **Flow-tools.** Tool set for working with NetFlow data URL: http://code.google.com/p/flow-tools/

УДК 004.4+519.68

В. В. Палагин, аспирант, Московский государственный университет приборостроения и информатики,
e-mail: art.stritn@gmail.com

К вопросу об ускорении параллельных программ для научно-технических расчетов за счет преобразования вложенных циклов

Рассматриваются вопросы, связанные с увеличением быстродействия многопроцессорных вычислительных систем архитектуры MPP (Massive Parallel Processing) путем преобразования вложенных циклов. Рассматриваются сопутствующие задачи научного и технического характера, приводятся результаты экспериментов.

Ключевые слова: *параллельные вычисления, многопроцессорные вычислительные системы, разработка параллельных программ, языки программирования сверхвысокого уровня, преобразование вложенных циклов*

Одним из вариантов решения национально значимых задач, требующих значительных вычислительных ресурсов, является объединение большого числа параллельно работающих процессоров в единую вычислительную систему. Основным вектором развития таких систем является повышение их производительности, которая зависит от быстродействия аппаратной платформы и скорости выполнения программного обеспечения. По этой причине разработка эффективных параллельных программ представляет собой одно из приоритетных и в то же время сложных для реализации направлений научных исследований.

В данной работе под целевой платформой для запуска параллельных программ будем подразумевать вычислительный кластер (ВК) — подкласс многопроцессорных вычислительных систем (МВС) с распределенной памятью (*Massive Parallel Processing*). По данным на июнь 2012 г. ВК занимают 80 % от всего числа суперкомпьютеров из списка top500, оставшиеся 20 % приходятся на MPP-системы [1]. В общем случае системы с распределенной памятью представляют собой группу вычислительных узлов (ВУ), соединенных между собой коммуникационной средой. В зависимости от сложности задач и выделяемых для их решения финансовых средств ВУ могут быть созданы на базе высокопроизводительных серверов (Opteron/Xeon/NVidia Tesla) или специально (в зависимости от класса задачи) спроектированных уникальных технологических решений, а в качестве коммуникационной среды могут быть использованы высокоскоростные и низколатентные сетевые технологии (Myrinet/InfiniBand). В случае вычислительных кластеров, рассматриваемых в настоящей статье, в качестве ВУ выступают серийные компьютеры, связанные между собой сетью передачи данных (Fast/Gigabit Ethernet).

Логично предположить, что подобная представленной выше организация параллельных вычислений порождает значительную разницу во времени доступа к данным, расположенным локально и в памяти других ВУ. Такая диспропорция времени выполнения операций действительно существует, она получила название "стена памяти". Причина в том, что операции в оперативной памяти (ОП) ВУ происходят значительно медленнее, чем операции в регистрах процессора. Если же работать через единое адресное пространство с памятью многих ВУ, то добавляются еще тысячи тактов на прохождение коммуникационной среды.

Одним из путей преодоления отмеченных узких мест в организации работы с памятью является рациональное распределение массива исходных данных по ВУ и минимизация числа межпроцессорных пересы-

лок. Такого эффекта можно добиться за счет создания более эффективной параллельной программы (далее используется термин оптимизация) под архитектуру МВС с распределенной памятью. Здесь отдельно следует отметить, что вопрос глобальной оптимизации параллельных программ является весьма обширным и рассмотреть все потенциально возможные пути его решения в одной работе невозможно. В связи с этим введем некоторые ограничения на постановку задачи, решение которой рассматривается в настоящей работе.

В рамках исследования, результаты которого рассматриваются далее, под понятием параллельных программ для научно-технических расчетов понимаются программы, реализующие конечно-разностные или сеточные методы решения краевых задач. С точки зрения разработчика параллельных программ на Си/С++/Фортран специфика таких программ заключается в распределении времени выполнения, до 80 % которого занимают вложенные циклы [2]. По этой причине, а также в силу влияния порядка вложения циклов на распределение данных по ВУ, направление исследования в данной работе было сосредоточено на поиске оптимального порядка вложения циклов.

Для того чтобы оптимизировать параллельную программу по времени выполнения, необходимо определить состав компонентов и выявить зависимости. Общее время выполнения параллельной программы можно представить как:

$$t_{glob} = \Sigma t_{loc} + t_{ext},$$

где t_{loc} — время выполнения одного вложенного цикла, t_{ext} — время выполнения программного кода за пределами циклов. По причине того, что работа с программным кодом за пределами циклов выходит за рамки данной статьи, эта величина принята за константу. Тогда задача минимизации времени выполнения параллельной программы сводится к задаче минимизации времени выполнения вложенных циклов, которое зависит от ряда параметров:

$$t_{loc} = f(m, c_{it}, p_{loop}),$$

где m — размер входных данных; c_{it} — общее число итерации цикла; p_{loop} — общая трудоемкость тела цикла. На основе оценки трудоемкости выполнения тела цикла можно определить общую трудоемкость выполнения вложенного цикла:

$$P_{it} = ((p_{loop}c_{it}) + P_{ext}) - (p_{loop}c_{br}),$$

где P_{ext} — общая трудоемкость выражений на разных уровнях вложенности цикла; c_{br} — число невыполненных итераций цикла вследствие досрочного выхода из цикла по условию или при срабатывании оператора безусловного перехода. В случае если досрочного выхода из цикла не произошло и он совершил максимальное число итераций, то последняя компонента в виде произведения трудоемкости тела цикла и числа невыполненных итераций равна нулю. В случае если между уровнями вложения цикла отсутствует исходный код, то значение P_{ext} будет равно нулю. Таким образом, в простейшем случае общая трудоемкость выполнения вложенного цикла будет определяться как произведение трудоемкости тела цикла и числа итераций. В данной работе общую трудоемкость вложенного цикла предлагается представить как сумму вычислительной трудоемкости и трудоемкости доступа к данным. Вычислительная трудоемкость в этом случае будет определена на основе числа тактов процессора, затраченных на выполнение арифметических операций, а трудоемкость доступа к данным — на основе тактов, затраченных на доступ к элементам массива данных в памяти ВК. Исходя из этого можно предположить, что после оптимизации параллельной программы ее ускорение будет существенным только в том случае, если вычислительная трудоемкость много меньше трудоемкости доступа к данным.

С точки зрения распределения массива исходных данных по ВУ кластерной системы, решение задачи нахождения оптимального порядка вложения циклов может быть найдено, в том числе, с помощью специализированных языковых средств разработки параллельных программ. При выполнении исследования, результаты которого представлены далее, в качестве удобного инструментария для работы с циклами использовался декларативный язык программирования сверхвысокого уровня "НОРМА" (Непроцедурное Описание Разностных Моделей Алгоритмов), который был разработан в Институте прикладной математики им. М. В. Келдыша (ИПМ РАН [3, 4]). Данный язык программирования ориентирован, прежде всего, на решение задач научно-технического характера, в том числе и расчетов методом конечных разностей (МКР), которые относятся к классу сеточных методов приближенного решения краевых задач. В качестве входных данных синтезатор принимает специально оформленный исходный код, а в качестве выходных данных разработчик получает исходный код на языках Си/Фортран со вставками вызовов одной из поддерживаемых библиотек передачи сообщений.

Существует несколько причин, по которым в качестве инструментария для модификации порядка вложения циклов предлагается использовать язык сверхвысокого уровня. Одной из таких причин является поддержка ряда конструктивных особенностей, присущих функциональному подходу в программировании. К их числу относятся такие как выполнение операторов независимо от порядка их расположения, отсутствие побочных эффектов вычислений, упор на описание правил вычисления, а не на подробную конкретизацию алгоритма. Эти свойства в комплексе уменьшают трудоемкость написания параллельных

программ и позволяют разработчику записывать расчетные формулы в математической форме в практически неизменном виде. Но основной причиной является наличие удобных механизмов модификации порядка вложения циклов и специализированного оператора для вычислений арифметических значений величин.

Модификация порядка вложения циклов происходит за счет изменения описания индексов распределения (*distribution index*). Данное описание влияет на распределение массива исходных данных и управление между ВУ, а также, если это необходимо, позволяет автоматически генерировать операторы обмена данными между ними. Распределению подлежат величины, участвующие в расчетах и имеющие индексы, совпадающие с теми, которые указаны в описании областей. Текущая реализация содержит два специализированных оператора для описания вычислительных действий, основной из которых предназначен для вычислений арифметических значений величин (*assume*). Данный оператор, по сути, является коллективным оператором, который при трансляции в результирующую программу на Си/Фортран развертывается в циклы. Такие коллективные операторы выступают в роли строительных блоков, инкапсулирующих в себе большую часть трудоемкости и смысловой нагрузки параллельной программы. Здесь необходимо пояснить особенности построения параллельных программ на языке сверхвысокого уровня. В качестве смыслового разделителя исходного кода могут применяться функции или разделы. Раздел представляет собой аналог функции, в котором можно статически задать распределение вычислений по ВУ целевой платформы. Если такое распределение не задано, то все вычисления выполняются на одном ВУ. Как правило, реальные программы состоят из набора распределенных разделов. В каждом из них выполняется минимально-возможный смысловой блок действий, включающий в себя несколько коллективных операторов. Такая организация структуры параллельной программы позволяет добиться компромисса между сохранением контроля над распределением вычислений со стороны разработчика и объемом/сложностью исходного кода.

Таким образом, задача нахождения оптимального распределения данных в каждом разделе параллельной программы сводится к задаче лексического анализа исходного кода. Цель выполнения такого рода анализа заключается в поиске всех коллективных операторов в рамках одного раздела и определение оптимального значения индексов распределения. В свою очередь модификация индексов распределения в параллельной программе на языке сверхвысокого уровня приведет к перестановке набора вложенных циклов в результирующей программе на Си/Фортран.

Определение оптимального набора индексов для каждого раздела исходной программы проводится на основе значения трудоемкости выполнения каждого коллективного оператора с учетом задействованных индексов. На данном этапе исследований значение трудоемкости вычисляется исходя из числа базовых арифметических операций в выражении. В процессе исследования был разработан специализированный транслятор для выполнения лексического анализа и модификации исходного кода. Из всего набора индексов распределения транслятор присваивает каждому варианту свой вес, который определяется в зависимости от трудоемкости каждого коллективного оператора. Выходными данными для транслятора является оптимизированный исходный код, из которого в дальнейшем генерируется результирующая параллельная программа на Си/Фортран с измененным порядком вложения циклов.

Для того чтобы на практике оценить реальное влияние порядка вложения циклов на время выполнения параллельной программы был проведен ряд практических экспериментов. В качестве тестовой программы использовалась программа умножения квадратных матриц размерностью 2000 элементов по ленточному алгоритму. В данном алгоритме исходные матрицы разбивают на горизонтальные и вертикальные полосы, которые копируются по всему массиву ВУ. Перемножение полос приводит к получению части блоков результирующей матрицы. По причине того, что объем памяти, необходимый для одновременного хранения исходных матриц целиком на каждом ВУ может превысить доступный объем физической памяти ВУ, вычисления организуются таким образом, чтобы в каждый момент времени ВУ содержали лишь часть элементов исходных матриц, а доступ к остальной части обеспечивался при помощи передачи сообщений. В качестве аппаратной платформы использовался вычислительный кластер кафедры ИТ-4 Московского государственного университета приборостроения и информатики. Результаты эксперимента представлены на рис. 1, см. вторую сторону обложки.

Для наглядности результаты с близкими значениями были объединены в три группы (*A*, *B*, *C*), где версии параллельных программ в группах обозначены индексами распределения в соответствии с терминологией языка сверхвысокого уровня. После преодоления порога в 12 ВУ время выполнения начинает возрастать для групп *B* и *C*, которые объединяют результаты для параллельных программ со значением оператора распределения данных *ji*, *jk* и *ij*, *ik* соответственно. Такое поведение стало результатом уменьшения размера блока передаваемых данных между ВУ вследствие того, что размерность исходной задачи осталась прежней, а число ВУ увеличилось.

Рассмотрим, каким образом происходит распределение массива исходных данных по ВУ кластерной системы для группы *A* со значением индексов $k = 1...10$, $i = 1$, где они используются для описания пе-

ременных на областях. Данное выражение означает, что все переменные, определенные на областях с индексами k и i, будут распределены по сетке ВУ с виртуальными номерами строк k в столбце с номером i. По причине ограниченной поддержки распределения вычислений по сетке ВУ со стороны использованного синтезатора, второй индекс i, определяющий номер столбца, может принимать только одно значение. Таким образом, переменные на областях с индексом k будут распределены на ВУ. Соответственно, в результирующей программе на Си/Фортран порядок вложенных циклов будет таким, что самым внутренним циклом будет цикл с индексом k. Операторы обмена данными между ВУ также будут сгенерированы на основе разбиения исходных матриц по этому индексу.

Кроме тестовой программы умножения матриц, работа построенного в ходе исследования транслятора была проверена еще на ряде параллельных программ, также выполняющих различные действия с матрицами. Результаты этой проверки представлены на рис. 2, см. вторую сторону обложки.

В целом положительная тенденция сохраняется, и версии параллельных программ с оптимальным порядком вложения циклов почти вдвое быстрее. Здесь следует отметить, что программы для научно-технических расчетов большого объема имеют весьма сложную внутреннюю структуру, что непременно отразится на возможностях оптимизации. Кроме того, специфика реализации конечно-разностных методов решения краевых задач в некоторых случаях может сделать невозможным перестановку циклов без глобального изменения исходного кода параллельной программы. Однако, несмотря на подобные ограничения, методы оптимизации, реализованные в системе правил лексического анализатора, могут использоваться при разработке новых эффективных параллельных программ для научно-технических расчетов, а также в специализированных математических библиотеках и шаблонах на языке программирования C++. Вместе с тем необходимо отметить, что исследования в данной области еще не завершены и в ближайшее время предстоит поэтапное тестирование разработанной системы правил (и ее программной реализации в виде специализированного транслятора исходного кода) на различных задачах в других предметных областях, анализ и обобщение полученных результатов.

Список литературы

1. **Статистика** 500 самых мощных компьютеров мира [Электронный ресурс] / TOP500 Statistics — Электрон. дан. URL: http://i.top500.org/stats свободный. Загл. с экрана.

2. **Воеводин В. В., Воеводин Вл. В.** Параллельные вычисления. СПб.: БХВ-Петербург, 2002. 608 с.

3. **Спецификация** языка НОРМА [Электронный ресурс] — URL: http://www.keldysh.ru/pages/norma/specif/ book1.htm, свободный. Загл. с экрана.

4. **Андрианов А. Н., Бугеря А. Б., Ефимкин К. Н.** НОРМА — специализированная система параллельного программирования. Томск: Изд-во Томского университета, 2002. 70 с.

ИНФОРМАЦИЯ

26—27 апреля в г. Санкт-Петербург пройдет
Тринадцатая Международная конференция в области обеспечения качества ПО
"Software Quality Assurance Days"

Предыдущая 12-я конференция проходила в г. Минск, где ее участниками стали более 550 профессионалов. Организатором традиционно выступает компания "Лаборатория тестирования".

Конференция охватит широкий спектр профессиональных вопросов в области обеспечения качества, ключевыми из которых являются:

- методики и инструменты тестирования ПО;
- автоматизация тестирования ПО;
- подготовка, обучение и управление командами тестировщиков;
- процессы обеспечения качества в компании;
- управление тестированием и аутсорсинг;
- совершенствование процессов тестирования и инновации.

Для многих специалистов и руководителей "Software Quality Assurance Days" это реальная возможность заявить о себе, повысить профессиональный уровень сотрудников, которые отвечают за ПО. SQA Days — это замечательная платформа общения и обмена опытом для людей, вовлеченных в сферу тестирования ПО. Ведущие профессионалы смогут рассказать о своих достижениях, показать, как эффективно использовать инструменты, методики и методологии. Для начинающих — это отличный шанс приобрести новые полезные знакомства в профессиональной среде.

Сайт конференции: http://it-conf.ru/ru/content/578.htm

УДК 004.032.26

И.С. Колбин, аспирант, Московский авиационный институт
(национальный исследовательский университет),
e-mail: iskolbin@gmail.com

Программный комплекс для решения задач математического моделирования с использованием нейросетевой методологии

Представлена структура разработанного автором программного комплекса для решения задач математического моделирования широкого класса физических объектов с использованием нейросетевой методологии. Подробно анализируются особенности реализации отдельных его составляющих. На примере решения тестовой задачи продемонстрированы этапы функционирования программного комплекса.

***Ключевые слова**: программный комплекс, численные методы, нейросетевое моделирование, бессеточные методы, нормализованные радальнобазисные сети, НРБС*

Введение

В настоящее время наметился интерес к применению нейросетевых технологий для решения задач математического моделирования. Построение математической модели сложных физических объектов с распределенными параметрами обычно сопряжено с составлением систем дифференциальных уравнений, заданием начальных и краевых условий, отношений, описывающих законы сохранения энергии и др. Отношения, описывающие состояние объекта, могут иметь нелинейный характер, содержать неточно заданные коэффициенты, использовать измерения, проведенные с некоторой погрешностью, что создает дополнительные сложности при поиске решения. Применение нейросетевых технологий позволяет в значительной степени преодолеть перечисленные сложности, оставаясь при этом в рамках единой методологии, без внесения значительных модификаций в унифицированные алгоритмы [1, 2]. Это обстоятельство порождает потребность в создании высокопроизводительных программных комплексов, реализующих такие подходы. Целью работы, результаты которой представлены в настоящей статье, является разработка эффективных алгоритмов и программного комплекса для решения задач математического моделирования с использованием нейросетевых технологий.

Постановка задачи и метод решения

В качестве отправной для описания модели объекта рассматривается краевая задача в достаточно общей постановке (представленной в [3]). В области Ω евклидова пространства требуется найти решение $u = u(\mathbf{x})$ уравнения

$$A(u) = f(\mathbf{x}),\ \mathbf{x} \in \Omega,$$

удовлетворяющее условию на границе Γ

$$B(u) = g(\mathbf{x}),\ \mathbf{x} \in \Gamma,$$

где $\mathbf{x}$ — входной вектор; A, B — интегро-дифференциальные операторы; f, g — некоторые заданные функции. Операторы могут быть нелинейными, содержать разрывы, менять тип в подобластях и обладать другими особенностями. Специальных требований к границе не предъявляется. Для учета начальных и краевых условий временная компонента либо заносится в вектор $\mathbf{x}$ (начальное условие включается в границу), либо используется гибридный метод с конечно-разностным разбиением по времени и нейросетевой аппроксимацией на каждом временном слое.

Решение основного уравнения ищется в виде нейросетевой модели. Метод решения заключается в подборе параметров нейросетевой аппроксимации. Обоз-

начим через $\tilde{u}(\mathbf{x}, \boldsymbol{\psi})$ выход сети (или, в общем случае, коллектива сетей), где $\boldsymbol{\psi}$— вектор параметров модели. Одним из возможных способов поиска параметров является их подбор через оптимизацию квадратичного функционала ошибки [3]. Несомненным достоинством этого способа является его универсальность (например, в сравнении с вариационным функционалом [1]), что позволяет применять единую методологию для широкого круга задач.

Функционал ошибки формируется из квадратов невязок при подстановке нейросетевой модели в исходное уравнение и граничные условия:

$$J(\tilde{u}) = \int_{\Omega} w_{\Omega}(A(\tilde{u}(\mathbf{x}, \boldsymbol{\psi})) - f(\mathbf{x}))^2 d\mathbf{x} + \\ + \int_{\Gamma} w_{\Gamma}(B(\tilde{u}(\mathbf{x}, \boldsymbol{\psi})) - g(\mathbf{x}))^2 d\mathbf{x},$$

где w_{Ω}, w_{Γ} — некоторые коэффициенты, которые "выравнивают" вклад внутренней и граничной составляющей относительно друг друга. Подбор этих параметров во многом зависит от условий задачи. Как правило, вычислить интегралы аналитически достаточно сложно или принципиально невозможно. По этой причине используется дискретный вариант функционала:

$$J(\tilde{u}) = \sum_{j=1}^{m_{\Omega}} w_{\Omega}(A(\tilde{u}(\mathbf{x}_j^{\Omega}, \boldsymbol{\psi})) - f(\mathbf{x}_j))^2 + \\ + \sum_{j=1}^{m_{\Gamma}} w_{\Gamma}(B(\tilde{u}(\mathbf{x}_j^{\Gamma}, \boldsymbol{\psi})) - g(\mathbf{x}_j))^2,$$

где m_{Ω}, m_{Γ} — число контрольных точек в области и на границе соответственно, $\mathbf{x}_j^{\Omega} \in \Omega$ — набор контрольных точек в области, $\mathbf{x}_j^{\Gamma} \in \Gamma$ — набор точек на границе. Выбор числа точек и характер их размещения (равномерный по области, более насыщенная концентрация в областях с ожидаемыми большими ошибками и т.д.) зависит от сложности решаемой задачи.

В качестве нейросетевой модели для получения результатов, которые используются в настоящей работе, использованы сети нормализованных радиально-базисных функций (НРБС) [4, 5]:

$$\tilde{u}\left(\mathbf{x}|\omega_i, \sigma_i, \mathbf{x}_i^0\right) = \frac{\sum_{i=1}^{n} \omega_i \varphi\left(\mathbf{x}, \sigma_i, \mathbf{x}_i^0\right)}{\sum_{i=1}^{n} \varphi\left(\mathbf{x}, \sigma_i, \mathbf{x}_i^0\right)},$$

где φ — заданная радиально-базисная функция; ω, σ, $\mathbf{x}^0 \subset \boldsymbol{\psi}$— параметры нейросетевой модели: веса, ширины и координаты центров соответственно. Подбор параметров осуществляется с использованием методов оптимизации.

Описание структуры программного комплекса

Разработанный автором и представленный в настоящей работе программный комплекс состоит из трех модулей, которые условно можно назвать *препроцессор*, *решатель* и *постпроцессор*. Общая схема программного комплекса представлена на рис. 1.

Рис. 1. Структура программного комплекса

В связи с тем, что упомянутые модули решают значительно отличающиеся друг от друга задачи как в смысле выполняемых ими функций, так и в плане вычислительной сложности их реализации, представляется целесообразным выбирать для каждого модуля соответствующие им языки программирования.

Препроцессор

Основной функцией препроцессора является преобразование пользовательских входных данных в формат, пригодный для обработки решателем. Так как подобная обработка проходит один раз для каждой задачи, то скорость работы препроцессора, вообще говоря, не является критичной. Значительно более важным представляется гибкость в его настройке и простота отладки. Особенно ценными при выборе языка для реализации препроцессора являются развитые возможности обработки строковых запросов. Оптимальным выбором представляются языки программирования высокого уровня, такие как Python, Lua, Perl, Ruby и др. В настоящей работе использовался язык Python.

Решатель

В функции решателя входит считывание данных в определенном формате, собственно поиск решения основной задачи и вывод данных в "сыром" виде (в виде вектора параметров аппроксимирующей модели ψ). Решатель является самым затратным компонентом комплекса в смысле использования им вычислительных ресурсов. Представляется разумным при реализации уделять внимание, прежде всего, высокой производительности. В качестве языков реализации представляется естественным выбрать C/C++ или Fortran. Причина в том, что кроме высокопроизводительного объектного кода данные языки предоставляют развернутую базу готовых научных библиотек для различного рода вычислений. В работе использовался язык программирования C.

Для решения задачи решатель должен получить на входе:

— набор условий, каждое из которых включает в себя функцию, вычисляющую значение функционала в точке условия, число контрольных точек и правило распределения контрольных точек;

— набор параметров нейросетевой модели: для каждой сети задается тип (нормализованная или нет), базисная функция, число нейроэлементов, порядок вычисляемой производной для каждого из условий;

— процедуру для вычисления начального приближения (в простейшем случае — случайные значения параметров в заданных интервалах);

— опционально задаваемую функцию для вычисления ошибки полученных решений (если известно аналитическое решение задачи или численное, полученное каким-либо другим методом).

Более детально следует рассмотреть первый пункт с точки зрения непосредственной реализации на алгоритмических языках. В простейшем случае можно непосредственно запрограммировать некоторый набор условий, а также позволить пользователю выбирать условия из этого набора и изменять коэффициенты в зависимости от задачи. Предлагаемый подход очень прост в реализации, однако недостаточно гибок, так как не представляется возможным заранее предугадать требуемый набор условий для произвольной задачи. Значительно больший интерес представляет такое построение решателя, при котором условие может целиком задаваться пользователем. Именно в этом случае непосредственно проявляется вся технологичность рассматриваемых методов, их адаптируемость к различным задачам без внесения модификаций в сами алгоритмы.

Рассмотрим перечисленные далее несколько типовых подходов к программной реализации решения задачи.

1. ***Полная перекомпиляция всего решателя под конкретную задачу.*** Достоинствами подобного подхода являются простота в реализации и высокая производительность, так как все процедуры проходят через оптимизирующий компилятор, который в значительной степени снижает накладные расходы на вызовы функций. Очевидным недостатком этого подхода является неудобство для пользователя и медленный старт.

2. ***Использование встраиваемых интерпретаторов,*** например, Python, Lua (строго говоря, эти языки являются не интерпретируемыми, а компилируемыми в байт-коды виртуальных машин) и т.д. Достоинствами такого решения являются удобство для пользователя и быстрый старт. К сожалению, интерпретация, как правило, выполняется достаточно медленно, что в значительной степени замедляет работу всего решателя. Следует отметить, что существует большое число проектов, целью которых является создание высокопроизводительных JIT-компиляторов (*Just-in-time*, осуществляют перевод в объектный код во время выполнения программы) для перечисленных языков. Особняком в их ряду стоит проект LuaJIT [6]. Разработанный в его рамках компилятор позволяет добиться сопоставимой с обычными AOT-компиляторами (*Ahead-of-time*, осуществляют перевод в машинный код перед началом выполнения программы) производительности без каких-либо ограничений в гибкости разработки.

3. ***Применение динамических библиотек.*** При таком подходе полученная производительность будет несколько ниже, чем при использовании подхода 1, вследствие накладных расходов, связанных с динамической компоновкой. Однако удобство для пользователя очевидно — нет необходимости перекомпилировать весь решатель. Из недостатков можно отметить, что данный подход является платформозависимым, так как реализация механизма динамической загрузки библиотек, вообще говоря, на разных системах отличается (а некоторые и вовсе не поддерживают данную технологию). К числу аналогичных по методам реализации проектам можно отнести TinyC [7], в рамках которого представляется компилятор, позволяющий проводить компиляцию функций "на лету". Следует

однако заметить, что этот компилятор работает на ограниченном наборе платформ.

Как следует из изложенного выше, все подходы обладают своими достоинствами и недостатками. В материалах, представленных в настоящей работе, использовался первый из перечисленных выше подходов. Его применение обусловлено тем обстоятельством, что решатель достаточно компактен и время перекомпиляции незначительно, что, в целом, нивелирует основной недостаток — медленный старт.

Решатель можно условно разбить на несколько блоков, а именно — блок вычисления значения и градиента функционала ошибки; блок оптимизации; блок методов расчета (прямой и гибридный). Взаимодействие между блоками следующее: метод использует процедуры оптимизации, которые, в свою очередь, вызывают вычисления значений и градиента функционала. Далее перечисленные блоки будут рассмотрены более подробно.

Блок вычисления значения и градиента функционала. Рассмотрим блок вычисления значения и градиента функционала ошибки. Непосредственно вычисление включает в себя несколько вложенных циклов, которые представлены на рис. 2.

Для вычисления значений и производных конкретного типа сети предварительно необходимо задать функции, вычисляющие значения и производные соответствующих базисных функций нейроэлементов, которые образуют сеть. Отметим, что не представляет сложности реализовать радиально-базисные сети совместно с нормализованными радиально-базисными сетями. Расчет самих сетей, вообще говоря, значительно проще вычислений базисов, которые проводятся независимо.

Необходимо отметить, что значительное внимание при разработке решателя было уделено промежуточному сохранению результатов, так как множество вычислений при расчете частных производных высших порядков базисных функций повторяется. Сохранение результатов расчетов позволяет в значительной степени увеличить быстродействие методов, но асимптотически сильно повышает затраты памяти. Данный недостаток, в определенном смысле, сглаживается компактностью нейросетевых представлений.

Рис. 2. Циклы, входящие в вычисление функционала

Рассмотрим циклы, начиная с самого нижнего. В процессе реализации *цикла по нейронам* вычисляются значения и производные базисной функции в соответствующей точке для каждого нейроэлемента. В рассматриваемом комплексе реализован расчет производной первого и второго порядка (включая смешанные производные). Производные более высоких порядков можно получить, например, через комбинацию сетей. В ходе *цикла по сетям* происходит вычисление выходов конкретных нейронных сетей (в простейших случаях в расчете участвует только одна сеть). В ходе выполнения *цикла по точкам* происходит вычисление суммы выходов сетей и их производных для всех точек, ассоциированных с условием. В результате реализации *цикла по условиям* происходит взвешенное сложение сумм, вычисленных в ходе выполнения цикла по точкам для каждого из условий.

Блок оптимизации. В блоке содержится набор процедур для оптимизации. В их числе методы нулевого порядка: метод конфигураций; метод деформируемого многогранника; метод генетических алгоритмов. Включены оптимизационные алгоритмы первого порядка: метод градиентного спуска; методы сопряженных градиентов (Флетчера-Ривса, Полака-Ривьера-Поляка, Хегера-Ченга и д.р.); квазиньютоновские методы (Девидона-Флетчера-Пауэлла, Бройдена-Флетчера-Гольдфарба-Шанно, Бройдена-Флетчера-Гольдфарба-Шанно с ограниченной памятью) [10, 11]. Заметим, что при построении этого блока важно, чтобы реализованные оптимизационные процедуры обладали единообразной формой вызова.

Блок методов. В рассматриваемом блоке содержится набор методов, по которым ведется построение и настройка нейросетевых моделей. В программный комплекс включены прямой и гибридный методы.

Постпроцессор

Задачей постпроцессора является преобразование выходных данных, полученных от решателя, в удобный для восприятия и анализа вид. В частности, как правило, требуется представить выходные данные в виде графиков и диаграмм. Подобные задачи не являются ресурсозатратными в смысле вычислительной мощности, поэтому их разумно решать с использованием высокоуровневых языков и библиотек.

В качестве примера следует особенно выделить свободно распространяемый программный пакет SciPy [8] для языка программирования Python, который является эффективным средством для проведения научных расчетов. Данный пакет включает в себя полный набор библиотек, необходимых для решения задач линейной алгебры, оптимизации, численного интегрирования, обработки сигналов и др. В пакет включена библиотека для визуализации графиков и диаграмм matplotlib, которая использовалась для подготовки материалов настоящей работы.

Модельная задача

Для тестирования разработанного программного комплекса рассматривается решение простой стационарной краевой задачи. В квадратной области единичного размера требуется найти решение уравнения

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - \lambda^2 u = 0,$$

удовлетворяющее граничным условиям

$$u(0, y) = \mathrm{sh}\mu(1 - y),\ u(1, y) = -\frac{\mathrm{sh}\mu(1-y)}{\mathrm{sh}\mu},$$
$$u(x, 0) = \cos\pi x,\ u(x, 1) = 0.$$

Постоянные в уравнениях: $\lambda = 2$, $\mu = \sqrt{\lambda^2 + \pi^2} \approx 3{,}72$ (аналогично работе [9]). Аналитическое решение задачи известно и может использоваться для оценки точности полученных приближенных решений

$$u(x, y) = \cos\pi x \frac{\mathrm{sh}\mu(1-y)}{\mathrm{sh}\mu}.$$

В качестве базисной была выбрана функция Гаусса:

$$\varphi(\mathbf{x}, \sigma, \mathbf{x}^0) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}^0\|^2}{2\sigma^2}\right).$$

Для решения использовалось $m_\Omega = 80$ контрольных точек внутри области и $m_\Gamma = 80$ на границе (по 20 на каждой из сторон). Были рассмотрены сети из 4, 8, 12 и 16 нейронов. Подбор параметров осуществлялся с использованием метода сопряженных градиентов CG_DESCENT[10].

Погрешность полученных решений и значения функционала в конце решения задачи для сетей различного размера

Результаты вычислений	Число нейронов n			
	4	8	12	16
Среднеквадратичные отклонения ε	0,01411	0,00119	0,00056	$7{,}19 \cdot 10^{-5}$
Значение функционала J	0,00233	0,00044	$5{,}63 \cdot 10^{-6}$	$1{,}03 \cdot 10^{-6}$

Для оценки точности полученных решений проведен расчет среднеквадратичных ошибок в узлах сетки:

$$\varepsilon = \sqrt{\frac{\sum_{i=0}^{N}\sum_{j=0}^{N}[u(hi, hj) - \tilde{u}(hi, hj)]^2}{(N+1)^2}},$$
$$N = 100,\ h = \frac{1}{N},$$

Результаты вычислений (среднеквадратичные ошибки ε, а также значения функционалов J в конце решения задачи) представлены в таблице.

На рис. 3 представлены решения в различных сечениях $y = \mathrm{const}$: аналитическое (точками); аппроксимированное НРБС из четырех (пунктирные линии) и из восьми нейронов (сплошные линии).

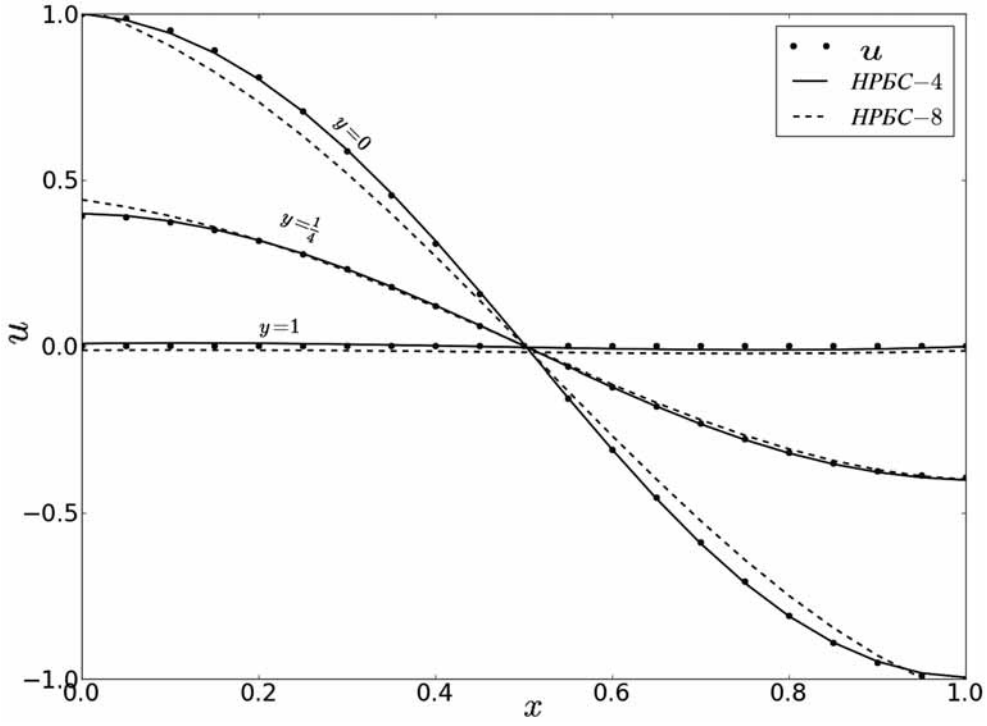


Рис. 3. Аналитическое (*u*) и приближенные НРБС решения

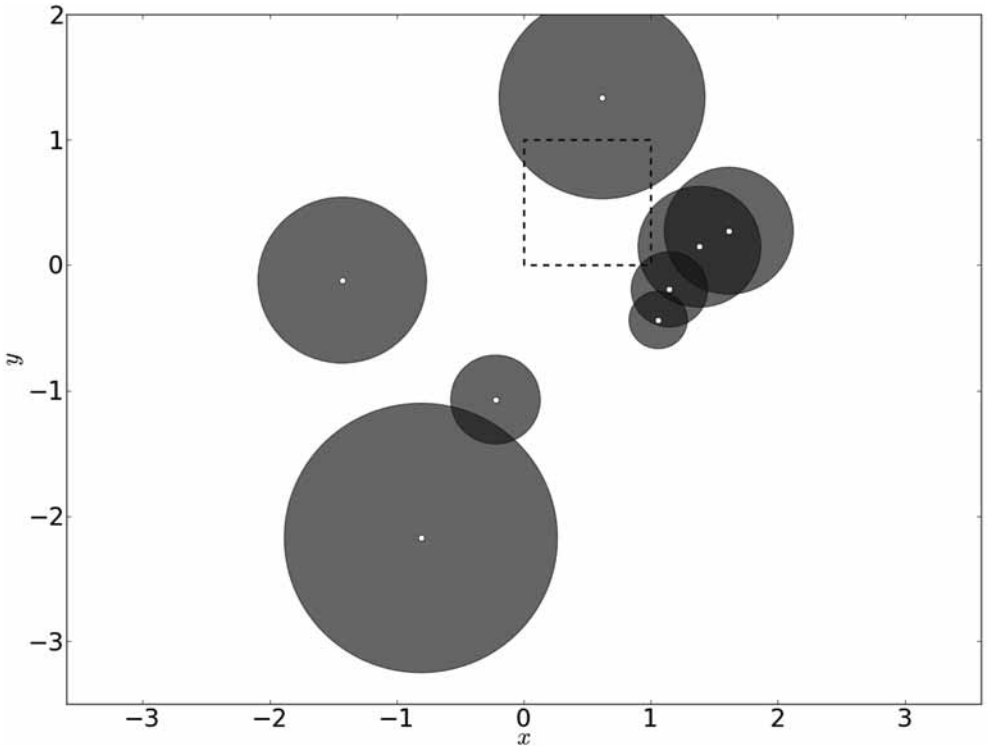


Рис. 4. Финальная конфигурация элементов нейронной сети

На рис. 3 видно, что решение, полученное с помощью сети из восьми нейронов, практически совпадает с аналитическим, что позволяет говорить о высокой точности метода. На рис. 4 показано размещение центров нейроэлементов на заключительной стадии подбора параметров. Радиусы окружностей соответствуют ширинам нейронов. Пунктиром выделена граница области задачи.

Следует отметить, что в результате подбора параметров модели ни один из центров нейроэлементов (белые точки на кругах на рис. 4) не расположился внутри области. Напротив, некоторые нейроны расположились на достаточно большом удалении. Тем не менее, вклад каждого нейрона в выход сети внутри области (и, соответственно, в результирующее решение задачи) существенен вследствие нелокального характера нормализованных РБС при использовании в качестве базисной гауссовой функции (что является важным отличием НРБС от классических РБС).

Заключение

В работе рассмотрен программный комплекс для математического моделирования широкого круга физических объектов с использованием нейросетевой методологии. Проанализированы особенности построения отдельных его компонентов. На примере решения модельной краевой задачи продемонстрирована работоспособность комплекса.

Список литературы

1. **Васильев А. Н., Тархов Д. А.** Нейросетевое моделирование. Принципы, алгоритмы, приложения. СПб.: Изд-во Политехнического университета, 2009.

2. **Колбин И. С., Ревизников Д. Л.** Решение задач математической физики с использованием нормализованных радиально-базисных нейроподобных сетей // Нейрокомпьютеры: разработка, применение. 2012. № 2. С. 12—19.

3. **Васильев А. Н., Осипов В. П., Тархов Д. А.** Унифицированный процесс моделирования физико-технических объектов с распределенными параметрами // Научно-технические ведомости СПбГПУ. Физико-математические науки. 2010. № 3. С. 39—51.

4. **Хайкин С.** Нейронные сети: полный курс, 2-е издание: Пер. с англ. М.: Вильямс, 2006. 394 с.

5. **Bugmann G.** Normalized Radial Basis Function Networks // Neurocomputing (Special Issue on Radial Basis Function Networks). 1998. Vol. 20. P. 97—110.

6. **The LuaJIT** Project. URL: http://www.luajit.org

7. **Tiny C** Compiler. URL: http://www.scipy.org

8. **Scientific** tools for Python. URL: http://www.bellard.org/tcc

9. **Chai Z., Shi B.** A novel lattice Boltzmann model for the Poisson equation // Applied Mathematics and Mechanics. 2008. Vol. 32. P. 2050—2058.

10. **Hager W., Zhang H.** A new conjugate gradient method with guaranteed descent and an efficient line search // SIAM Journal on Optimization. 2005. Vol. 16. P. 170—192.

11. **Kelley C. T.** Iterative Methods for Optimization. Philadelphia: Society for Industrial and Applied Mathematics, 1999.

УДК 004.021

Д. Л. Ревизников, д-р физ.-мат. наук, проф.,e-mail: reviznikov@inbox.ru,
С. А. Семенов, аспирант, e-mail: stdx@inbox.ru**,** Московский авиационный институт (национальный исследовательский университет)

Особенности молекулярно-динамического моделирования наносистем на графических процессорах

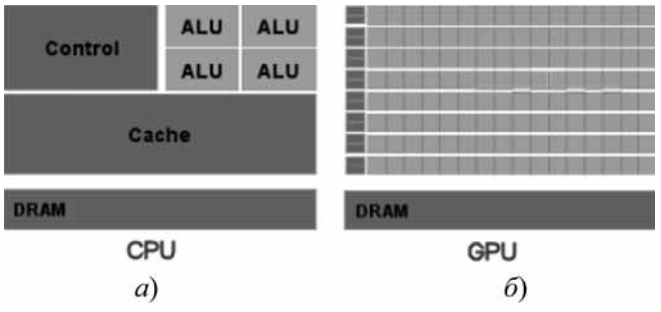
Рассматриваются вопросы распараллеливания вычислений на графических процессорах применительно к задачам молекулярно-динамического моделирования наноструктур. Обсуждаются особенности программной реализации алгоритмов и дается описание способов повышения производительности программ. Представленные результаты свидетельствуют о высокой эффективности использования параллельных вычислений на графических процессорах в задачах рассматриваемого класса.

Ключевые слова: *параллельные вычисления, графические процессоры, молекулярно-динамическое моделирование, наноматериалы, углеродные наноструктуры*

В последнее время все больший интерес вызывает проведение массивных вычислений с использованием графических ускорителей. Известны варианты применения графических процессоров к задачам линейной алгебры, математической физики, вычислительной аэрогидродинамики [1—4].

В отличие от современных универсальных центральных процессоров (CPU), видеочипы предназначены для параллельных вычислений с большим числом арифметических операций. Большее число транзисторов графических процессоров (GPU) работает по прямому назначению — обработке массивов данных, а не управляет исполнением (*flow control*) немногочисленных последовательных вычислительных потоков. Представленное выше назначение наглядно иллюстрирует сравнение схем загрузки CPU и GPU (рис. 1). В загрузке CPU небольшое число ядер — четыре арифметическо-логических устройства (ALU), блок управления (Control), устройство кэширования (Cache), оперативная память (DRAM). В загрузке GPU присутствуют большое число ядер, соответствующее им число блоков управления и кэширования, достаточное количество общей памяти.

С начала 2000-х гг. GPU вычисляли цвета пикселей на экране с помощью программируемых арифметических устройств, пиксельных шейдеров (*pixel shader*). Пиксельный шейдер получает на входе координаты (x, y) точки на экране и дополнительную информацию, а на выходе должен выдать конечный цвет этой точки. В качестве дополнительной информации могут выступать входные цвета, текстурные координаты или иные атрибуты, передаваемые шейдеру на этапе его выполнения. Так как арифметические действия над входными цветами и текстурами контролируются программистом, то в качестве данных могут выступать не только цвета. В ноябре 2006 г. nVidia выпустила серию видеокарт GeForce на основе архитектуры CUDA. В отличие от предыдущих поколений GPU, в которых вычислительные потоки подразделялись на вершинные и пиксельные шейдеры, в архитектуру CUDA включен унифицированный шейдерный конвейер, позволяющий программе общего назначения задействовать любое арифметическо-логическое



а) *б*)

Рис. 1. Графическое представление вычислителей в CPU (*а*) и GPU (*б*)

устройство (ALU). Поскольку nVidia рассчитывала, что новое семейство графических процессоров будет использоваться для вычислений общего назначения, то ALU были сконструированы с учетом требований IEEE к арифметическим операциям над числами с плавающей точкой одинарной точности. Кроме того, был разработан набор команд, ориентированный на вычисления общего назначения, а не только на графику. Устройствам GPU разрешен произвольный доступ к памяти для чтения и записи, а также доступ к программно-управляемой кэш-памяти. Эти средства были добавлены в архитектуру CUDA с целью создать GPU, который отлично справлялся бы с вычислениями общего назначения.

В качестве примера приведем самую доступную видеокарту, одну из самых первых с поддержкой CUDA GeForce 8600 GTS для моделирования. Она имеет следующие характеристики: память GDDR3 256 Мбайт, четыре мультипроцессора по восемь потоков, частоту памяти 1050 МГц, пропускную способность шины 128 бит. Это значит, что видеокарта способна выполнять 32 шага цикла вычислений за один такт. Если функция в цикле не приводит к рассинхронизации вычислительных потоков, то это дает преимущество в 32 раза по сравнению с одним ядром процессора.

Архитектура вычислителей CUDA состоит из параллельных экземпляров ядра. Эти параллельные экземпляры называются вычислительными блоками. Исполняющая среда CUDA позволяет расщепить блоки на вычислительные потоки (*thread*). Запуск вычислительного ядра (функции) выглядит следующим образом:

```
//numBlocks, numThreads
function<<<N,1>>>(dev_a, dev_b,dev_c);
```

где `numBlocks`, `numTreads` — число вычислительных блоков и потоков соответственно, `dev_a`, `dev_b`, `dev_c` — некоторые аргументы, передаваемые функции. Второй параметр в угловых скобках указывает, сколько вычислительных потоков создать для каждого блока. В этом примере программист конфигурирует *N* блоков, состоящих из одного вычислительного потока, всего получается *N* параллельных вычислительных потоков. Язык CUDA C предоставляет в распоряжение программы разделяемую память (*shared memory*). Компилятор CUDA C создает копию разделяемых переменных (*__shared__*) в каждом блоке, и все вычислительные потоки используют их совместно. Буферы разделяемой памяти находятся в самих GPU, а не в общей памяти (DRAM — динамическое запоминающее устройство с произвольной выборкой). Таким образом, задержки при доступе к разделяемой памяти существенно меньше, чем к глобальной памяти. Чтобы вычислительные потоки взаимодействовали, необходим механизм синхронизации. Например, если вычислительный поток *A* записывает какое-либо значение в разделяемую память, то вычислительный поток *B*, использующий это значение, не должен приступать к работе, пока вычислительный поток *A* не завершил операцию.

Помимо глобальной и разделяемой памяти nVidia предоставляет и другие виды памяти. Константная память имеет ограничения по сравнению с глобальной, но большую производительность. Если обращения к константной памяти из всех потоков происходят к одним и тем же данным только для чтения, то трафик между памятью и процессором уменьшается за счет трансляции результатов операции чтения на половину вычислительного блока (*half warp*) и за счет кэша памяти на кристалле. Доступ к памяти является узким местом во многих алгоритмах, поэтому наличие механизма, позволяющего улучшить производительность доступа, необходимо для эффективных вычислений.

Одной из наиболее перспективных сфер приложения графических процессоров является молекулярно-динамическое моделирование. При этом существенно то, что в силу ограниченности радиуса взаимодействия между частицами здесь возможно практически полное распараллеливание процессов, происходящих в различных областях пространства.

Молекулярно-динамическое моделирование основано на решении уравнений движения совокупности частиц:

$$m_i \frac{d^2 r_i}{dt^2} = F_i = -\nabla_i E(r_1, r_2, ..., r_N), \qquad (1)$$

где m_i — масса i-й частицы (всего N частиц); r_i — радиус-вектор; F_i — сила, действующая на i-ю частицу; $\nabla_i E(r_1, r_2, ..., r_N)$ — градиент потенциальной энергии.

В настоящей работе уравнение (1) аппроксимируется конечно-разностной схемой скоростей Верле второго порядка точности:

$$r_i(t + \Delta t) = r_i(t) + \Delta t v_i(t) + \frac{1}{2}\Delta t^2 a_i(t),$$

$$a_i(t + \Delta t) = \frac{F_i}{m_i},$$

$$v_i(t + \Delta t) = v_i(t) + \frac{1}{2}\Delta t(a_i(t) + a_i(t + \Delta t)),$$

где v_i — скорость i-й частицы; a_i — ускорение.

В данной схеме необходимо сохранять координаты, скорости, ускорения каждого атома. Этот объем данных записывается в глобальную память графической карты. Моделирование проводится на видеокартах nVidia, так как фирма предоставляет удобные инструменты для программирования. Число вычислительных потоков выбирается по аппаратному размеру вычислительного блока (*warp*) видеокарты, а число этих блоков — таким образом, чтобы расчет каждой частицы шел в отдельном вычислительном потоке, и пустых потоков было наименьшее количество.

Для определения действующей на атом результирующей силы, вообще говоря, необходимо рассчитать вклады, вносимые всеми атомами системы. Однако структура большинства потенциалов межчастичного взаимодействия позволяет учитывать не все атомы, а лишь соседние по отношению к рассматриваемому атому. Таким образом, у атома возникает определен-

ный набор соседей, для которых необходимо провести вычисление сил. Здесь можно выделить два основных способа [5]: составление списка соседей или разбиение пространства на ячейки. В первом случае с использованием ограничивающих сфер выбираются все соседние частицы, которые вносят вклад в результирующую силу. При этом необходимо обновлять данный список на каждом шаге и для каждого атома.

Согласно второму способу пространство разбивается на ячейки, определяемые ограничивающей функцией (рис. 2). В этом случае учитываются дополнительные атомы, которые практически не вносят вклад в результирующую силу. Однако данный подход обладает важным достоинством — конфигурация ячеек известна заранее и не изменяется в процессе моделирования.

Применительно к распараллеливанию на графических процессорах более эффективным оказывается использование ячеистой модели движения. При этом для частиц, находящихся в некоторой ячейке, можно учитывать только взаимодействие с частицами из соседних ячеек. Для определения соответствия частицы определенной ячейке можно использовать не трехмерные координаты, а хэш-функцию:

$$hash_i = z_i \cdot YSize \cdot XSize + y_i \cdot XSize + x_i,$$

где *Xsize*, *Ysize*— размеры ячеек (в данной реализации в ангстремах).

Процесс моделирования разбивается на несколько шагов, которые видеокарта может обработать без перезагрузки данных и рассинхронизации:

1) интегрирование уравнений движения частиц;

2) определение хэш-функций;

3) сортировка атомов;

4) разделение атомов по ячейкам;

5) определение коэффициентов потенциала межчастичного взаимодействия;

6) расчет потенциала взаимодействия, сил, ускорения.

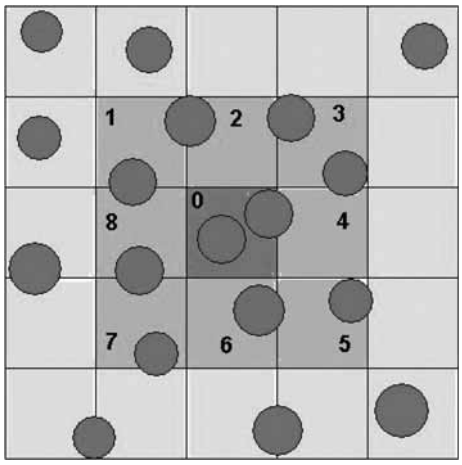


Рис. 2. Ячеистая модель движения, соседние ячейки пронумерованы одним из способов

Рассмотрим перечисленные пункты более подробно. На шаге интегрирования выполняется одна операция:

$$r_i(t + \Delta t) = r_i(t) + \Delta t v_i(t) + \frac{1}{2}\Delta t^2 a_i(t).$$

Данные, необходимые для одного шага одного вычислительного потока, располагаются в глобальной памяти видеокарты, потому что нужно хранить координаты, скорости, ускорения всех атомов.

Параметры же потенциала копируются в разделяемую память (*shared*) каждого вычислительного блока.

В программе интегрирование запускается следующей строкой [6]:

```
thrust::for_each(
   thrust::make_zip_iterator(thrust::make_
tuple(d_pos4, d_vel4, d_acel4)),
   thrust::make_zip_iterator(thrust::make_
tuple(d_pos4+numParticles, d_vel4+numParti-
cles, d_acel4+numParticles)),
   integrate_functor(deltaTime));
```

где `deltaTime` — шаг интегрирования по времени; `d_pos4` — массив координат атомов; `d_vel4` — массив скоростей атомов; `d_acel4` — массив ускорений атомов; `numParticles` — число атомов.

При таком запуске компилятор сам выделяет нужную память и определяет необходимое число потоков вычисления.

Процедура определения хэш-функций запускается следующей строкой [6]:

```
// execute the kernel
  calcHashD<<< numBlocks, numThreads>>>
(gridParticleHash,
    gridParticleIndex,
    (float4 *) pos,
    numParticles);
```

где `gridParticleHash` — результат, вычисляемый хэш-функцией; `gridParticleIndex` — номер атома; `pos` — координаты атома.

Для сортировки используется параллельный алгоритм *thrust sort* [7].

В статье рассматривается моделирование углеродных структур, которые весьма перспективны, так как обладают рядом уникальных свойств [8]. Математическое моделирование таких структур связано с высокими вычислительными затратами в силу сложной структуры потенциала взаимодействия.

Для моделирования углеродных наноструктур используется потенциал Бреннера [8], который состоит из притягивающей $V^A(r_{ij})$ и отталкивающей $V^R(r_{ij})$ компонент, а также функции порядка связи b_{ij}:

$$E^B = \sum_i \sum_{j>i} (V^R(r_{ij}) - b_{ij} V^A(r_{ij})),$$

где $r_{ij} = |r_i - r_j|$ — расстояние между i-м и j-м атомами.

Первая сумма хорошо соответствует архитектуре графических вычислителей. Для вычисления вто-

рой суммы, рассматривается ограничивающая функция:

$$f^c = \begin{cases} 1, r_{ij} < D_{ij}^{\min} \\ \frac{1}{2}\left(1 + \cos\frac{\pi\left(r_{ij} - D_{ij}^{\min}\right)}{D_{ij}^{\max} - D_{ij}^{\min}}\right), \\ D_{ij}^{\min} < r_{ij} < D_{ij}^{\max} \\ 0, r_{ij} > D_{ij}^{\max} \end{cases},$$

где $D_{ij}^{\max}$ — максимальное расстояние действия потенциала; $D_{ij}^{\min}$ — расстояние сглаживания действия потенциала. Ограничивающая функция определяет расстояние действия потенциала.

Таким образом, рассматриваются только атомы, находящиеся в пределах $D_{ij}^{\max}$. Это позволяет более эффективно использовать ячеистую модель движения. Каждый атом находится в определенной ячейке, так что при соответствующем подборе размеров ячеек все соседние атомы попадут в ту же ячейку или соседние. Так как использование хэш-функции позволяет свести трехмерные координаты в один массив, то соседние атомы в этом массиве будут располагаться рядом. Функция сортировки позволяет определить границы ячеек в этом массиве. Функция порядка связи включает в себя все особенности соединения атомов: тип связи, число атомов, угловое вращение, смещение двугранных углов, табличные значения экспериментальных констант отдельных атомов. После расчета значений этой функции остается запустить вычислители еще раз для получения результирующей силы, действующей на атом.

Для вывода моделируемой системы на заданную температуру и получение равновесного расположения атомов применяется специальная процедура термостатирования [10].

Графический процессор быстро обрабатывает простые функции и медленно вычисляет сложные. Поэтому расчет правых частей уравнений движения разбивается на два этапа: определение функции порядка связи и расчет сил, ускорений, скоростей.

Вычисление функции порядка связи и ее производных запускается следующими строками программы:

```
coeff<<<numParticles,
dim3(3,3,3)>>>((float4*)sortedPos,
// numBlocks, numThreads
              cellStart,
              cellEnd,
              numParticles,
              (float*)bondCoeff);
coeffDiff<<< numParticles,
dim3(3,3,3)>>>((float4*)sortedPos,
//numBlocks, numThreads
              cellStart,
              cellEnd,
              numParticles,
              (float3*)bondDiff);
```

где `cellStart` —индекс начала ячейки; `cellEnd` — индекс конца ячейки; `bondCoeff` — значения необходимых функций порядка связи, `bondDiff` — значения производных функций порядка связи. Здесь конфигурируется запуск видеокарты для расчета трехмерного расположения атомов: `dim3(3,3,3)` определяет, что рассматриваются непосредственно соседние ячейки, а первый параметр определяет, что число блоков вычисления равно числу атомов. Данная конфигурация выбрана для nVidia GeForce GTX 480 как наиболее эффективная. Отметим, что для GeForce 8600 GTS использовались другие параметры конфигурации: 32 — для размера блока и numParticles/32 — для размера вычислительной сети видеокарты.

Вычисление новых скоростей запускается следующим образом:

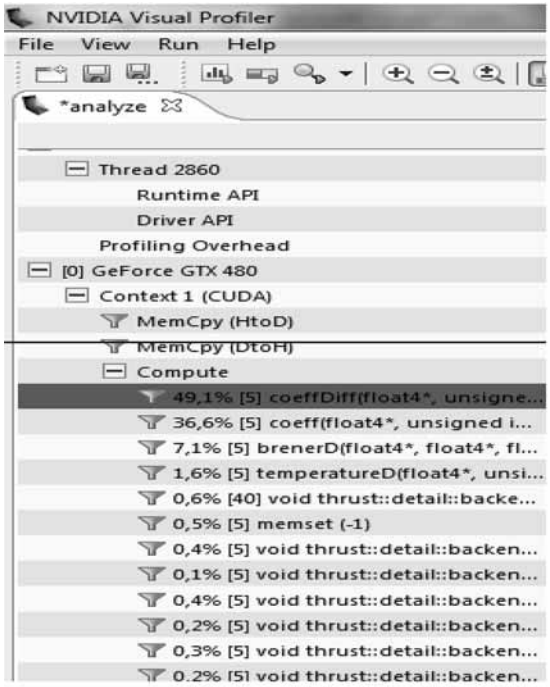
```
brenerD<<< numBlocks, numThreads>>>
    ((float4*)newVel,
    (float4*)newAcel,
    (float4*)sortedPos,
    (float4*)sortedVel,
    (float4*)sortedAcel,
    gridParticleIndex,
    cellStart,
    cellEnd,
    numParticles,
    deltaTime,
    Upot,
    (float*)bondCoeff,
    (float3*)bondDiff);
```

где `newVel` — вычисленные скорости; `newAcel` — вычисленные ускорения; `sortedPos` — координаты на предыдущем шаге вычисления в сортированном порядке; `sortedVel` — скорости на предыдущем шаге, `sortedAcel` — ускорения на предыдущем шаге; `Upot` — потенциальная энергия системы атомов; `numTreads`=32; `numBlocks=numParticles/numTreads` выбраны как наиболее эффективные из множества протестированных вариантов.

Для тестирования программного обеспечения проводилось моделирование ряда углеродных наноструктур. Рассматривались графеновые листы, фуллерены и нанотрубки (рис. 3, см. третью сторону обложки).

Производительность программы исследовалась специальной утилитой nVidia — Visual Profiler. На рис. 4 показан результат измерения времени работы функций программы. Наибольшее время занимает вычисление коэффициентов производных функции порядка связи, а всего для вычисления потенциала тратится 185 мкс на каждом шаге. При продолжительном расчете необходимо три минуты компьютерного времени для молекулярно-динамического моделирования одной наносекунды существования углеродной наноструктуры. Отметим, что при этом стабилизируются кинетическая и потенциальная энергии, а результаты расчета хорошо согласуются с данными лабораторных исследований [11, 12].

В целом полученные результаты свидетельствуют о высокой эффективности использования параллельных алгоритмов на графических процессорах в задачах молекулярно-динамического моделирования наноструктур.



Analysis | Details | Console | Settings

Name	Start Time	Duration	Grid Size	Block Size	Regs	Static SMem	Dynamic SMem	Size	Throughput
coeffDiff(float4*, unsigned i...	153,107 ms	1,3 ms	[258,1,1]	[3,3,3]	59	432	0	n/a	n/a
coeffDiff(float4*, unsigned i...	146,851 ms	1,294 ms	[258,1,1]	[3,3,3]	59	432	0	n/a	n/a
coeffDiff(float4*, unsigned i...	157,776 ms	1,282 ms	[258,1,1]	[3,3,3]	59	432	0	n/a	n/a
coeffDiff(float4*, unsigned i...	162,574 ms	1,272 ms	[258,1,1]	[3,3,3]	59	432	0	n/a	n/a
coeffDiff(float4*, unsigned i...	167,815 ms	1,237 ms	[258,1,1]	[3,3,3]	59	432	0	n/a	n/a
coeff(float4*, unsigned int*,...	156,795 ms	978,832 µs	[258,1,1]	[3,3,3]	48	400	0	n/a	n/a
coeff(float4*, unsigned int*,...	161,624 ms	948,735 µs	[258,1,1]	[3,3,3]	48	400	0	n/a	n/a
coeff(float4*, unsigned int*,...	152,159 ms	946,614 µs	[258,1,1]	[3,3,3]	48	400	0	n/a	n/a
coeff(float4*, unsigned int*,...	145,903 ms	945,846 µs	[258,1,1]	[3,3,3]	48	400	0	n/a	n/a
coeff(float4*, unsigned int*,...	166,877 ms	936,276 µs	[258,1,1]	[3,3,3]	48	400	0	n/a	n/a
brenerD(float4*, float4*, flo...	154,586 ms	185,329 µs	[9,1,1]	[32,1,1]	62	400	0	n/a	n/a
brenerD(float4*, float4*, flo...	148,355 ms	185,296 µs	[9,1,1]	[32,1,1]	62	400	0	n/a	n/a

Рис. 4. Измерение времени работы функций утилитой nVidia — Visual Profiler

Список литературы

1. **Guoming H., Cuiping L., Hong C., Xiaoyong D., Haijun F.** Using Graphics Processors for High Performance SimRank Computation // Knowledge and Data Engineering. IEEE 2012. Vol. 24 N. 9. P. 1711—1725.

2. **Винников А. В., Винников В. В., Ревизников В. В.** Численное решение уравнений в частных производных с использованием графических ускорителей // Известия МГИУ. Информационные технологии. 2007, № 1 (6). С. 15—22.

3. **Fukazawa K., Umeda T.** Performance measurement of magnetohydrodynamic code for space plasma on typical scalar-type supercomputer systems with a large number of cores // International Journal of High Performance Computing Applications. 2012. August. P. 310—318.

4. **Che-Rung L., Zhi-Hung C., Quey-Liang K.** Parallelizing the Hamiltonian Computation in DQMC Simulations: Checkerboard Method for Sparse Matrix Exponentials on Multicore and GPU // IEEE 26th International Parallel and Distributed Processing Symposium. Shanghai, China. 2012. P. 1889—1897.

5. **Allen M. P.** Introduction to Molecular Dynamics Simulation Computational Soft Matter: From Synthetic Polymers to Proteins // Lecture Notes. John von Neumann Institute for Computing. NIC Series. 2004. Vol. 23. P. 1—28.

6. **Рекомендации** разработчиков nVidia. URL: http://developer.nvidia.com.

7. **Алгоритм** thrust. URL http://thrust.github.com/

8. **Endo M., Iijima M., Dresselhaus M. S.** Carbon nanotubes. PERGAMON. 1996.

9. **Brenner D. W., Shenderova O. A., Harrison J.A.** A second-generation reactive empirical bond order (REBO) potential energy expression for hydrocarbons // Journal of Physics: Condensed Matter. 2002. Vol. 14. P. 783—802.

10. **Hunenberger P. H.** Thermostat algorithms for molecular dynamics simulations // Advances in Polymer Science. 2005. Vol. 173. P. 105—149.

11. **Yin M. T., Cohen M. L.** Will Diamond Transform under Megabar Pressures? //Physical Review Letters. 1983. Vol. 50. N 25. P. 2006.

12. **Yin M. T., Cohen M. L.** Structural theory of graphite and graphitic silicon //Physical Review B. 1983. Vol. 29. N 12. P. 6996.

УДК 002.55

Ю. И. Петров, преподаватель, **Ю. В. Шупикова**, студент,
Московский государственный университет приборостроения и информатики,
e-mail:mailtoyuripetrov@gmail.com

Современные информационные системы нормоконтроля выпускных квалификационных работ

Приведен обзор информационных систем нормоконтроля выпускных квалификационных работ, представленных на рынке программного обеспечения, описаны их ключевые особенности. На основании выполненного сравнительного анализа их возможностей отмечена необходимость разработки собственного программного обеспечения, отвечающего современным требованиям и адаптированного под условия отечественной практики.

Ключевые слова: *информационная система, нормоконтроль, выпускная квалификационная работа, программное обеспечение*

Введение

Нормоконтроль выпускной квалификационной работы (ВКР) — неотъемлемый этап завершающей стадии обучения, через который должен пройти каждый выпускник высшего учебного заведения. Наличие в настоящее время большого числа документов, стандартов и положений делает данный процесс достаточно трудоемким как для сотрудников, занимающихся проверкой выпускных работ, так и для самих студентов, а в некоторых случаях и довольно затратным [1]. Это обуславливает актуальность разработки и использования в процессе проверки ВКР специальных информационных технологий и реализующих их программных средств, которые смогут повысить эффективность подготовки работы к итоговой защите.

Обзор существующих систем

В настоящее время на рынке программного обеспечения (ПО) представлено несколько систем, используемых в сфере подготовки ВКР и их нормоконтроля. На мировом рынке, по мнению авторов, заслуживают внимания две разработки этого назначения, а именно **StyleEase** и **docVerifier**.

ПО компании **StyleEase** предназначено для автоматического форматирования документов, выполненных в текстовом процессоре Microsoft Word, что позволяет "сосредоточиться на самом содержимом, а не на его оформлении" [2]. Программный продукт сопровождает оформление документа, начиная с титульной страницы и заканчивая списком источников. Приложение обладает следующими характеристиками:

- полностью совместимо с американскими стандартами APA [3], MLA [4], Chicago/Turabian [5], регламентирующими различное оформление академических работ (каждая версия распространяется отдельно);
- работает на PC и Mac (Word 2011);
- автоматически форматирует документ и список источников;
- включает бесплатный доступ к WorldCat — крупнейшей в мире библиографической базе данных, насчитывающей свыше 240 млн записей обо всех видах произведений на 470 языках мира [6]. Доступ с получением цитат и выдержек возможен непосредственно через приложение или с использованием стандартного веб-браузера;
- сохраняет каждую ссылку из списка источников в собственной базе данных для дальнейшего использования;
- запускается и проводит обработку целевого документа не более чем за 10 мин.

Продукт выполнен в виде надстройки Word — устанавливаемого компонента, который добавляет пользовательские команды и новые возможности к приложениям Microsoft Office, в частности Microsoft

Word. Надстройки могут быть применены для создания новых или обновления существующих функций различного типа, что позволяет повысить производительность пользователя [7].

Интерфейс ПО StyleEase (APA-версии) представлен на рис. 1.

Отличительной особенностью StyleEase является комплексная поддержка цитирования и работы со списком источников. Для добавления ссылки на источник достаточно вызвать соответствующий мастер (рис. 2). Его интерфейс, включающий список возможных типов источников (книга, интернет-источник, журнал и т. д.) и предлагающий заполнить запрашиваемые им поля, приведен на рис. 3.

Сопровождая процесс написания академической работы на каждом этапе, StyleEase поддерживает функцию печати документа. Перед выводом на принтер модуль, реализующий эту функцию, проверяет и при необходимости обновляет/синхронизует оглавление, рисунки и таблицы, подготавливая документ к итоговой печати. Программный продукт снабжен качественным справочным руководством, также на его сайте присутствуют обучающие видеоролики, предназна-

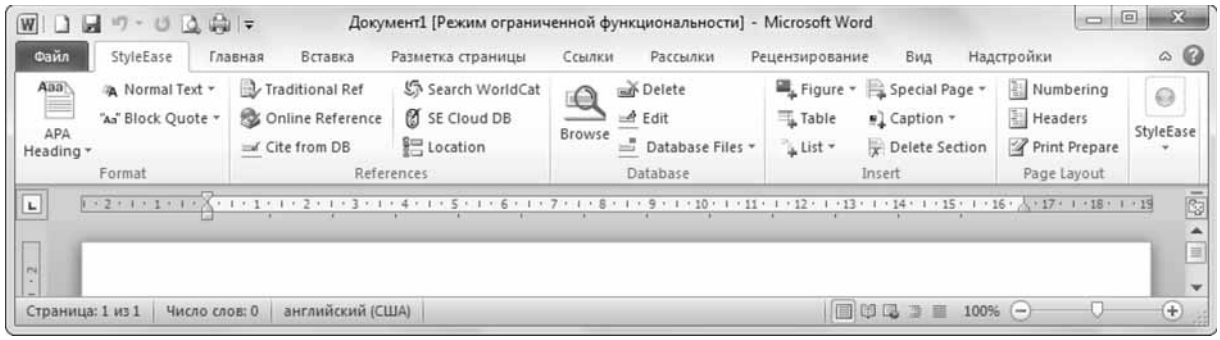


Рис. 1. Надстройка StyleEase for APA Style в Microsoft Word 2010

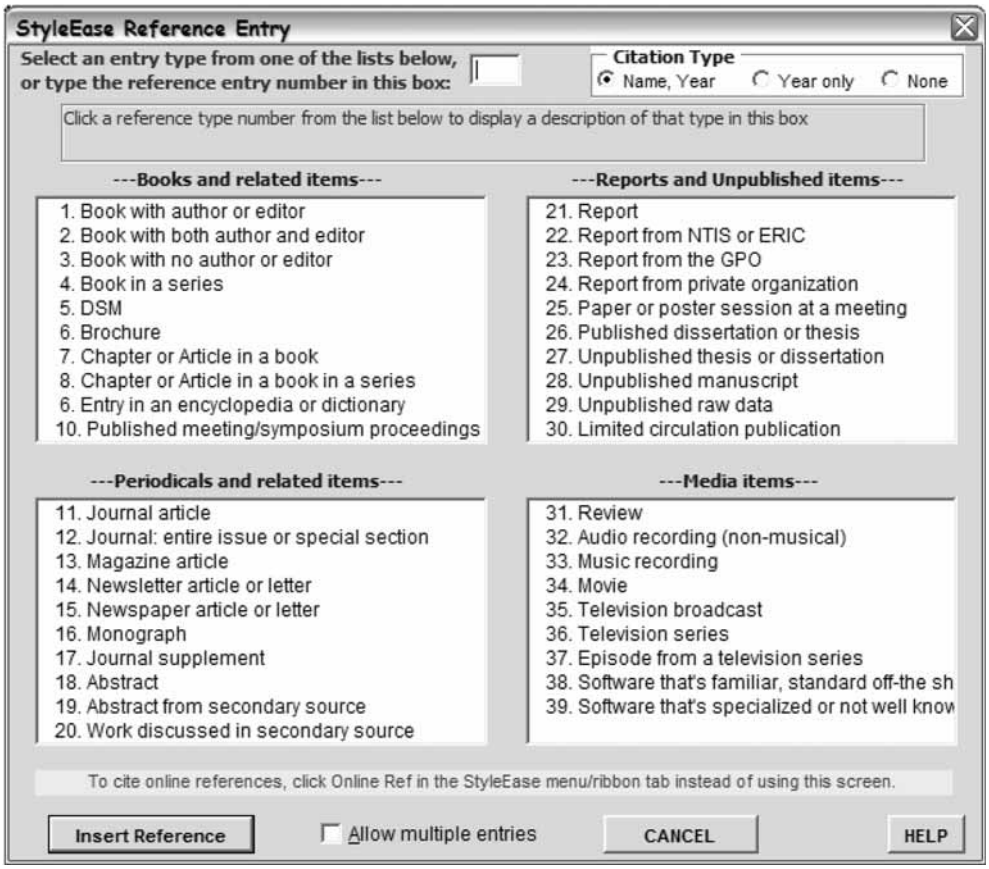


Рис. 2. Мастер добавления ссылки на источник в StyleEase for APA Style

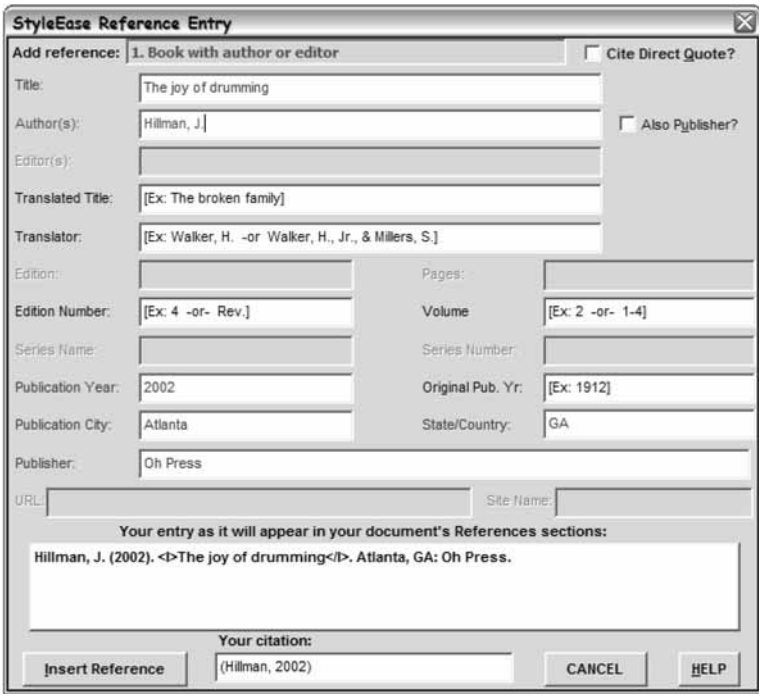


Рис. 3. Форма для заполнения информации об источнике для добавления ссылки в StyleEase for APA Style

Таблица 1

Стоимость различных редакций ПО StyleEase, долл. США

Способ получения	Редакция (операционная система Windows или Mac OS)		
	StyleEase for APA Style	StyleEase for MLA Style	StyleEase for Chicago/Turabian Style
CD	40	–	
Загрузка с веб-сайта	35		
CD + вторая лицензия	50	–	

Примечание: «–» — редакция не предоставляется данным способом.

ченные для быстрого освоения предоставляемой надстройки.

StyleEase является коммерческим ПО, цены на различные редакции продукта приведены в табл. 1. При покупке нескольких редакций приложения производитель предлагает гибкую систему скидок. Кроме того, потенциальному покупателю предоставляется возможность скачать бесплатную ознакомительную версию, в которой доступное для обработки число документов и пунктов списка источников ограничивается пятью элементами.

К основным недостаткам приложения следует отнести:

— заявленную разработчиками возможность работы только в английской версии редактора Microsoft Word, в частности, русскую версию офисного пакета данная надстройка не поддерживает;

— поддержку исключительно американских стандартов оформления документов (APA, MLA, Chicago/Turabian).

Указанные недостатки существенно ограничивают возможность применения надстройки в отечественной практике. Они ограничивают степень масштабируемости программного продукта, а языковое ограничение делает невозможным использование надстройки в пакетах Microsoft Word, не поддерживающих англоязычный интерфейс. Однако StyleEase работает со всеми версиями Microsoft Word 2000—2010 (за исключением "Starter"-редакций, не поддерживающих макросы), поддерживает Mac и, учитывая относительно небольшую стоимость копии, заслуживает внимания в качестве вспомогательного средства оформления документа.

Продукт **docVerifier** представляет собой веб-приложение, в отличие от StyleEase напрямую предназначенное для коррекции оформления готового документа — выпускной работы в режиме онлайн [8]. По заявлению авторов, проект предназначен:

- для выпускников вузов, а также персонала учебного заведения для оформления ВКР различного уровня;
- для преподавательского состава вузов в целях оформления их собственных публикаций;
- для форматирования больших документов с помощью Microsoft Word в крупных организациях.

Интерфейс главной страницы веб-сайта представлен на рис. 4.

Рис. 4. Главная страница приложения docVerifier

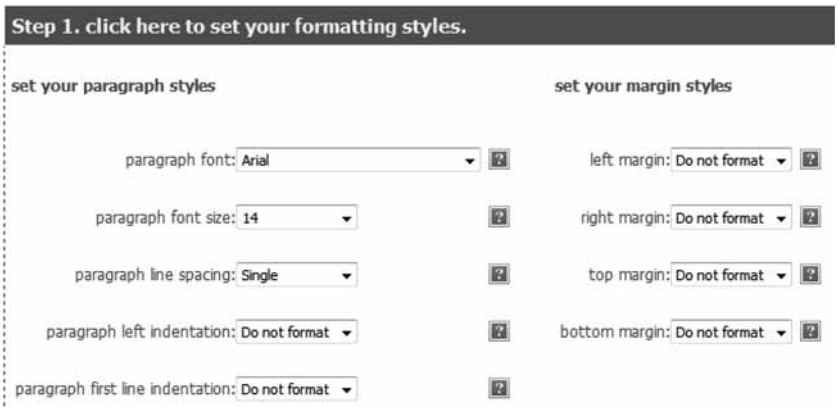


Рис. 5. Фрагмент блока установки стилей форматирования в docVerifier

Алгоритм работы приложения схематично указан на главной странице и описывается слоганом "upload-download-graduate" ("загрузи—скачай—окончи учебное заведение"). После нажатия на кнопку "Try it for Free" для отправки работы на проверку требуется выполнить следующие действия.

- Установить стили форматирования (рис. 5). Настройка сопровождается удобной подсказкой (пиктограмма-вопрос).
- Загрузить исходный документ для обработки (рис. 6).

В результате работы системы на указанный адрес электронной почты приходит письмо о завершении проверки со ссылками на два файла — полностью отформатированный и исходный документ с указанием сделанных исправлений в виде стандартных примечаний Microsoft Word.

Дополнительной особенностью системы является наличие на ее сайте авторизации для студентов вузов, с которыми существуют партнерские соглашения, что свидетельствует об определенном уровне зрелости данного продукта. Согласно информации, представленной на сайте, приложение docVerifier используется такими учебными заведениями, как Индианский университет в Блумингтоне, Университет Пердью, Университет Джорджа Мейсона, Университет Нотр-Дам, Университет Акрона (все — США).

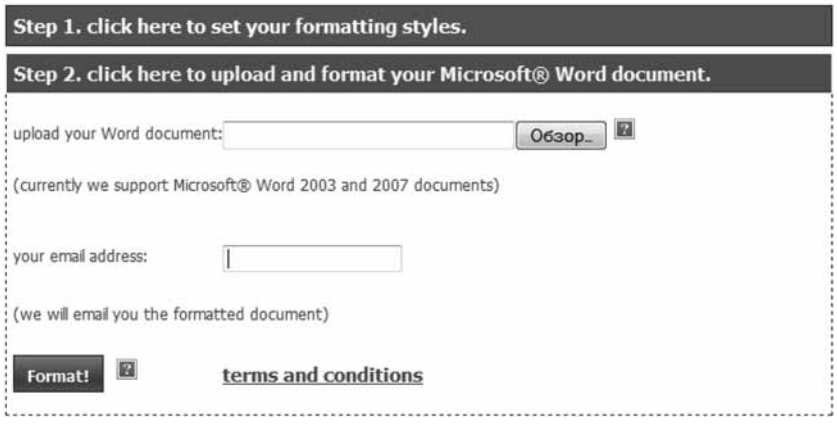


Рис. 6. Блок отправки исходного документа для форматирования в docVerifier

Приложение является кроссплатформенным (за счет веб-реализации) и бесплатным. В сочетании с представленными выше функциональными возможностями, данный программный продукт является эффективным средством валидации документов и заслуживает высокой оценки.

К функциональным недостаткам системы следует отнести недостаточно удобную настройку форматирования, а также то, что число контролируемых параметров документа достаточно мало чтобы можно было говорить о комплексной и гибкой проверке документа. Указанные параметры не представляется возможным сохранить, и при очередной отправке документа на проверку настройку необходимо проводить снова. Последнее обстоятельство указывает на необходимость доработки приложения. В частности, для технологии ASP.NET, на которой разработан проект, возможны различные варианты реализации отмеченных функций [9, 10]. Кроме того, на данный момент складывается впечатление, что работы над проектом приостанавливаются. Свидетельство тому то, что авторы недоступны ни по одному из указанных адресов электронной почты, последнее партнерское соглашение с вузами датируется 2009 г., обновление сайта — 2011 г.

На отечественном IT-рынке программные продукты подобного рода практически отсутствуют. Заслуживает внимания **пакет макросов "Методичка"**, представляющий собой набор инструментов для работы с документами Microsoft Word. По заявлениям авторов, "пакет упрощает обработку неформатированного текста, сканированной литературы, помогает быстро отформатировать документ, курсовую или дипломную работу, значительно экономит время при работе с потоком документации, автоматизируя рутинную однообразную работу" [11]. Приложение состоит из одного файла — шаблона Microsoft Word с встроенными макросами, одноименными вкладкой на ленте, панелью инструментов, встроенными стилями, командами клавиатуры и др. В случае, если необходимо использовать пакет в уже существующем документе, достаточно подключить к последнему предоставляемый разработчиком шаблон.

Интерфейс пакета (надстройки) приведен на рис. 7.

Функциональность пакета довольно обширна. К основным его возможностям следует отнести следующее.

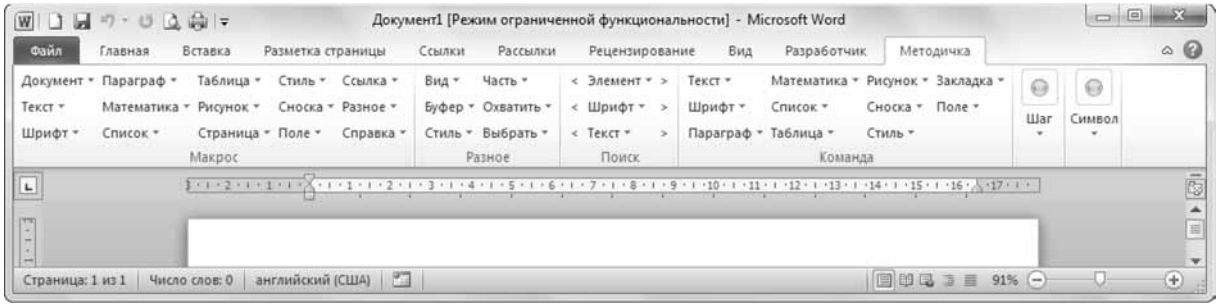


Рис. 7. Вкладка "Методичка" на ленте в Microsoft Word 2010

- *Работа с документом в целом:* полное форматирование документа (в том числе в автоматическом режиме); обработка элементов (как по отдельности, так в группе), включая страницы, таблицы (в том числе ячейки, строки, столбцы), рисунки, параграфы, строки, слова, закладки, сноски, поля и др.; пакетная обработка файлов и замена текста; разбивка/склеивание документов, импорт таблиц и рисунков.
- *Работа с текстом:* форматирование знаков препинания, сокращений и аббревиатур; выборка, поиск и замена текста (через стандартный диалог или с использованием регулярных выражений).
- *Работа с математическим аппаратом:* общее форматирование чисел и знаков математических операций; работа с дробными и целыми частями вещественных чисел; оформление и форматирование формул (в том числе проверка латиницы/кириллицы в тексте формулы).
- *Работа с параграфами:* удаление пустых абзацев, настройка отступов и интервалов, группировка и сглаживание окончаний параграфов.

Таблица 2

Сравнение продуктов StyleEase, docVerifier и «Методичка»

Показатель	Программный продукт		
	StyleEase	docVerifier	«Методичка»
Технические и товарные характеристики			
Тип приложения	Надстройка Word	Веб-приложение	Надстройка Word
Интерфейс	Вкладка на ленте (Word 2007 и выше), группа меню (Word 2003 и ниже)	Веб-интерфейс	Вкладка на ленте (Word 2007 и выше), группа меню (Word 2003 и ниже)
Язык интерфейса	Английский	Английский	Русский
Стоимость, долл. США	От 35	Бесплатно	От 5
Функциональные характеристики			
Упрощение основных операций форматирования документа	+	–	+
Проверка и коррекция оформления готового документа с текстом ВКР	–	+	+/–
Оформление списка источников	+	–	–
Ключевое предназначение	Полное сопровождение написания статей, выпускных работ, диссертаций	Проверка оформления готового документа с текстом выпускной работы	Автоматизация рутинных операций, помощь в форматировании документа
Ключевые недостатки	Поддержка только англоязычных версий Microsoft Word и американских стандартов оформления	Недостаточная гибкость приложения, «заброшенность» проекта	Перегруженность интерфейса, сложность комплексной проверки документа

Примечание: «+» — поддерживается полностью; «–» — не поддерживается; «+/–» — поддерживается частично.

- *Работа со шрифтами:* правка начертания; выравнивание, масштабирование и подгонка начертания шрифта.
- *Работа с таблицами:* форматирование размеров и положения таблиц; удаление пустых строк и ячеек; оформление подписей и примечаний таблиц.
- *Работа со списками:* форматирование списков в текстовом виде; преобразование и удаление списков и перечислений.
- *Работа с рисунками:* автоматическое оформление рисунков; масштабирование и удаление рисунков; перенос текста на полотно и преобразование надписей в текст; выравнивание рисунков по сетке.
- *Работа со стилями:* восстановление форматирования текста по его стилю; поиск и стилизация заголовков; стилизация элементов документа.
- *Прочее:* работа с закладками и гиперссылками; очистка колонтитулов и их распределение по документу; форматирование маркеров и сносок; сброс форматирования и структуры; стилизация параграфов в соответствии с их уровнем и т.д.

Стоимость пакета макросов составляет 10 долл. США, обновление с предыдущей версии — 5 долл. США. "Пожизненная" лицензия (включающая текущий пакет и все будущие обновления) предоставляется за 50 долл. США. Консультации по установке и использованию, а также исправление выявленных ошибок проводятся автором на бесплатной основе.

Набор функций пакета не имеет прямых аналогов в текстовом процессоре Word. В связи с этим, несомненно, данный пакет отвечает заявленным характеристикам. Кроме того, автор предоставляет исходный код макросов (на встроенном языке Office — VBA), что в случае необходимости позволяет расширить его функциональность. Имея при этом схожее с рассмотренным выше StyleEase предназначение, надстройка не позволяет выполнять такие обладающие определенной спецификой действия, как, например, ведение списка литературы. Это обстоятельство несколько снижает эффективность от ее использования при полноценном написании выпускных работ. Обзор функциональных возможностей приложения позволяет сделать вывод, что в большинстве случаев пакет будет служить лишь вспомогательным средством, сопровождающим процесс написания ВКР. Для комплексной проверки оформления работы потребуется, по крайней мере, сложная настройка пакета (выбор и настройка макросов) или написание дополнительных макросов для поддержки необходимых в этом случае функций.

Сравнительная характеристика рассмотренных программных продуктов приведена в табл. 2.

Заключение

Последние десятилетия связаны с резким ростом информатизации всех сфер деятельности. Модернизация коснулась и сферы образования — внедрение новых современных технологий можно наблюдать на всех образовательных ступенях. В свою очередь практически каждая ступень предусматривает написание письменных работ, где одной из главных является выпускная квалификационная работа.

В статье рассмотрены современные приложения, предназначенные для нормоконтроля ВКР. Проведенный обзор и сравнительный анализ показали, что ни одна из систем в полной мере не отвечает требованиям, предъявляемым к системам подобного типа [1]. Это обстоятельство указывает на актуальность разработки собственного ПО для проведения нормоконтроля ВКР на основе современных способов автоматизации текстового процессора Microsoft Word [12].

Список литературы

1. **Шупикова Ю. В., Петров Ю. И., Кунгурцева К. В., Гаранина Е. А.** Нормоконтроль выпускных квалификационных работ: сущность и необходимость автоматизации // Актуальные вопросы науки: Материалы VI Международной научно-практической конференции. Москва. 10.07.2012. М.: Спутник+, 2012. С. 64—71.

2. **APA Format** Software. Automatic Research Paper Formatting. StyleEase. StyleEase Software. URL: http://www.styleease.com/

3. **APA Style**. American Psychological Association. URL: http://www.apastyle.org/

4. **What** Is MLA Style? Modern Language Association. URL: http://www.mla.org/style/

5. **Chicago** Documentation Style. Board of Regents of the University of Wisconsin System. URL: http://writing.wisc.edu/Handbook/DocChicago.html

6. **WorldCat.org:** The World's Largest Library Catalog. WorldCat. URL: http://www.worldcat.org/

7. **Просмотр** и установка надстроек, а также управление ими в приложениях Office. Microsoft. URL: http://office.microsoft.com/ru-ru/word-help/HA010354315.aspx

8. **docVerifier**. Gosoft Solutions LLC. URL: http://docverifier.com/

9. **Общие** сведения об управлении состоянием ASP.NET. Microsoft. URL: http://msdn.microsoft.com/ru-ru/library/75x4ha6s.aspx

10. **Рекомендации** по управлению состоянием ASP.NET. Microsoft. URL: http://msdn.microsoft.com/ru-ru/library/z1hkazw7.aspx (дата обращения: 03.07.2012).

11. **МЕТОДИЧКА**.dot — Пакет макросов Word. URL: http://methodichka.ru/

12. **Петров Ю. И., Шупикова Ю. В., Викулина Д. А., Макаров С. Н.** Способы и средства автоматизации текстового процессора Microsoft Word // Перспективы развития информационных технологий: сборник материалов VIII Международной научно-практической конференции / Под общ. ред. С. С. Чернова. Новосибирск: ООО "Агентство «СИБПРИНТ»", 2012. С. 98—107.

УДК 004.4

В. В. Костюк, канд. техн. наук, доц., **М. А. Ушанов,** студент, Российский государственный университет инновационных технологий и предпринимательства,
e-mail: Viacheslav.Kostyuk@itbu.ru

К организации лабораторного практикума "Разработка программного обеспечения информационных систем на основе интеграции IBM-продуктов"

Данная статья отражает опыт, полученный авторами в процессе обучения студентов методам и средствам программирования на языке Java с использованием продуктов IBM. Освоение ими различных технологий программирования, конфигурирования и настройки средств разработки программных продуктов происходит в рамках лабораторного практикума. Анализируются вопросы, возникающие у студентов в ходе разработки программного обеспечения систем сбора и обработки, хранения и выдачи по запросам данных в информационных системах, а также подходы к их решению в рамках выполнения практикума.

Ключевые слова: *программирование, лабораторные работы, технологии программирования, интеграция IBM-продуктов, конфигурирование*

В лаборатории кафедры "Информационные системы" Российского государственного университета инновационных технологий и предпринимательства (РГУИТП) в рамках дисциплины "Информационные технологии" разработан и реализуется лабораторный практикум. Его целями являются:

— донесение до студентов базовых сведений о разработке программного обеспечения информационных систем разных уровней реализации, включая клиент-серверные разработки, толстого, тонкого клиента, web-реализации;

— получение студентами знаний и навыков применения современных [1] технологий Java-платформы.

Особенностью данного лабораторного практикума является использование средств разработки на основе продуктов компании IBM.

В рамках подготовки практикума ставилась и была решена задача обучения студентов созданию таких программных модулей, как навигаторы (меню), формы, отчеты на основе баз данных с применением технологий апплетов, сервлетов, XML, JSP, JSF, POJO, JPA, опираясь на спецификацию J2EE. Также студентам дали понять, как владеть библиотеками JDBC, средствами визуальных редакторов, разных моделей и шаблонов. Все лабораторные работы связаны единой целью создания автоматизированной системы ERP-ориентации. В результате освоения программы практикума студенты овладевают инструментальными средствами разработки от компании IBM и хорошо ориентируются в вопросах создания программного обеспечения информационных систем. Эти средства они используют для разработки программного обеспечения прикладных информационных систем с необходимыми для их сопровождения пользовательскими интерфейсами, видами разнообразной выдачи информации и управления ходом решения задач.

Для эффективного проведения занятий в рамках программы лабораторного практикума необходимо правильно сконфигурировать инструментальные средства среды разработки. Базовые средства в среде разработки программного обеспечения, которая рассматривается в контексте данной работы, включает три основных IBM-продукта:

— СУБД DB2 Express 9.7;

— среда разработки приложений Eclipse;

— WebSphere Application Server CE.

Для того чтобы эти средства представляли единую интегрированную [2] среду освоения лабораторного практикума, их необходимо дополнять другими, обеспечивающими интеграцию, технологическими компонентами. К ним относятся такие компоненты, как web-сервер (HTTP server), браузер FireFox. Кроме этого необходимы: виртуальная машина Java (VM), библиотеки Jdk1.6.22, Java Runtime Environment (JRE6), отдельные плагины, расширяющие возможности IDE Eclipse, драйвер СУБД DB2, а также ряд других модулей, которые упоминаются в лабораторных работах.

Все перечисленные выше факторы должны учитываться при инсталляции среды разработки перед началом выполнения лабораторного комплекса для обеспечения бесперебойной работы группы студентов. Для надежной установки этих средств как единой среды должен быть создан инсталляционный (конфигурационный) файл дистрибутивной загрузки такого интегрированного продукта. Этот файл должен не только установить все компоненты среды разработки, но и настроить их для реализации функций, необходимых для решения прикладных задач. Учитывая то обстоятельство, что упомянутые выше три основные IBM-продукта, используемые в лабораторном практикуме, распространяются свободно, каждый студент может устанавливать эти продукты и пользоваться ими в домашних условиях, перенося затем выполненные проекты в лабораторные условия. В этом случае возникают вопросы рассогласования сред выполнения практикума, о чем приходится напоминать студентам и пояснять причины возникающих трудностей. Необходимо отметить, что практикум выполняется в многопрофильной лаборатории, где работают студенты, выполняющие практические работы по другим дисциплинам, и они могут невольно нарушить файлы проектов, выполняемых в рамках рассматриваемого практикума.

С учетом перечисленных выше факторов, необходимы дополнительные меры организационного характера, чтобы обеспечить должный уровень преемственности результатов выполнения лабораторных работ, связанных между собой единым циклом освоения материала, а именно — переход от предыдущей работы к последующей, с прерыванием в одну и даже более недель. Эти меры реализуются путем копирования и хранения результатов выполнения каждой лабораторной работы на личные съемные носители самим студентом с тем, чтобы в случае сбоя можно было без потери информации продолжать работы после восстановления копий. Например, отлаженную базу данных по своей предметной области студенту следует хранить до конца выполнения всего лабораторного практикума, так как она используется в других лабораторных работах.

Конечно, не все лабораторные работы в рамках практикума в полной мере используют функциональные возможности, представляемые средой разработки [3]. Однако значительная их часть, особенно работы, связанные с клиент-серверной технологией, без полной настройки не могут быть выполнены. Рассмотрим, какие возможности интегрированной среды разработки необходимы и используются в каждой лабораторной работе. Одновременно представим лабораторный материал, который преподается студентам для практического освоения.

Лабораторная работа № 1 "Введение в СУБД IBM DB2". Целью этой работы является знакомство с сервером баз данных DB2 Express-C Community Edition и особенностями языка SQL.

В теоретической части этой работы студентам предлагается познакомиться с историей развития СУБД DB2, существующими версиями, их возможностями. Далее рассматриваются основные инструменты СУБД DB2: центр управления DB2 (DB2 Control Center), его главное окно; редактор команд DB2 (Command Editor), который используется для построения и выполнения запросов на языке SQL. Кроме результата запроса можно проанализировать план его выполнения. Рассматриваются возможности мастера создания баз данных, а также ассистента конфигурирования DB2, который помогает изменить конфигурацию как конкретной базы данных, так и всей СУБД DB2. Далее разбираются операторы DDL (Data Definition Language), предназначенные для определения структуры хранения данных в СУБД и операторы DML (Data Manipulation Language), предназначенные для управления данными. Достаточно подробно рассматриваются вопросы организации баз данных в DB2, а также логическая структура сервера DB2.

Выполняя практическую часть, студенты создают и работают с базой данных по индивидуальному заданию в рамках отдельной предметной области. Решение задач этой части работ предполагает использование таких возможностей, как создание триггеров ссылочной целостности, хранимые процедуры, резервное копирование базы данных и ее восстановление.

В данной лабораторной работе возникают вопросы, в основном связанные с расхождением имен схем баз данных. Такие имена в DB2 автоматически присваиваются по данным учетных записей пользователей и частый перенос базы данных с домашних компьютеров на лабораторные создает трудности выполнения заданий, тем более, что восстановление базы данных из сторонней резервной копии требует особого внимания. Этого конфигурационного минимума достаточно и для следующей лабораторной работы.

Лабораторная работа № 2 "Технология XML". Целью данной работы является освоение технологии XML и освоение технологии "pureXML" в СУБД IBM DB2.

В рамках этой работы рассматриваются следующие вопросы: история развития XML-технологии; основные конструкции языка XML [3]; что собой представляет правильно оформленный XML-документ, как избежать конфликтов имен; раскрытие понятия схемы XML-документа, ее конструктивных элементов; связывание этой схемы с документом (с приведением примеров).

Для практической реализации заданий осуществляется первое знакомство с продуктом Eclipse в минимальном для данной работы объеме знаний об Интегрированной среде разработки (Integrated Development Environment, IDE). Даются понятия проекта

(Project), рабочей области пользователя (Workspace). Работа с XML-технологией сводится к формированию в среде Eclipse XML-документа, рассмотренного в теоретической части лабораторной работы, включая создание схемы этого документа средствами визуального редактора, генерацию XML-документа, проверку его на правильность и наполнение документа данными. Студент получает задание, касающееся представления таблицы БД в виде XML-документа.

Лабораторная работа № 3 "Разработка Java-апплетов". Целью этой работы является освоение технологии Java Applet и среды разработки Eclipse.

В ходе выполнения этой работы объясняется понятие апплета, а также развертывание его в web-странице с использованием различных тегов. Далее рассматривается расширенное представление интегрированной среды разработки, содержащей необходимый набор инструментов IDE Eclipse для программирования. В их числе: текстовый редактор; компилятор или интерпретатор языка программирования; средства автоматизации сборки программы; средства отладки. Рассмотрены модульная архитектура IDE Eclipse, понятие перспектива (Perspective) как совокупность определенным образом расположенных окон (Views); понятие View (окно) — часть перспективы, отвечающая за выполнение конкретных функций. Расширяется понятие Project (проект) — логическая структура, включающая исходный код, библиотеки, а также различные файлы, необходимые разработчику. Далее подробно рассмотрены функции IDE Eclipse для работы с языком Java.

В практической части студенту предлагается запустить Eclipse, создать Java и web-проекты, выбрать HTTP-сервер (Apache HTTP Server), на котором будут публиковаться HTML-страницы, создать простейшие Java-апплеты (без и с графическим интерфейсами), а также апплет, который обращается к СУБД DB2 и формирует табличное представление результатов (рис. 1). В качестве базы данных используется база данных, созданная в лабораторной работе № 1. В качестве предметной области выбрана деятельность по управлению движением пригородных поездов.

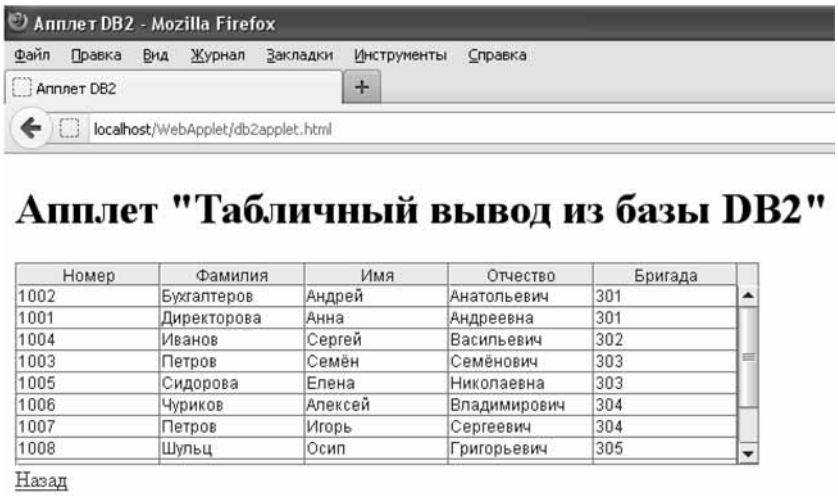


Рис. 1. Пример выдачи результатов Лабораторной работы № 3

Эта лабораторная работа требует дополнительной интеграции и конфигурирования таких составляющих среды ее выполнении как Apache HTTP Server, браузер Internet Explorer или FireFox, виртуальная машина Java (VM), библиотеки Jdk1.6.22, Java Runtime Environment (JRE6), визуальный конструктор (Visual Editor), драйвер СУБД DB2.

Даже при отлаженной конфигурационной инфраструктуре, при переносе данных на другой компьютер студент должен понимать, какие минимальные действия, восстанавливающие работоспособность лабораторной работы, необходимо выполнить. К их числу относятся: проверка наличия нужной базы данных; запуск HTTP-сервера; размещение и публикация на нем web-проекта; а также другие, в зависимости от ситуации.

Степени конфигурирования уровня лабораторной работы № 3 достаточно для выполнения следующей работы, не требующей серверной части приложений.

Лабораторная работа № 4 Разработка "толстого клиента". Целью этой работы является освоение технологии отдельного простого приложения с доступом к данным СУБД IBM DB2 без применения сервера приложений.

В теоретической части этой работы студенту предлагается разобраться с визуальными компонентами (beans) библиотеки Swing, включая JMenu, JMenuItem и JMenuBar. Практическая работа ограничивается созданием приложения, результатом работы которого является форма выдачи данных на базе двух связанных между собой таблиц.

В ходе выполнения работы студенту предлагается освоить стиль разработки, в основе которого лежит шаблон, состоящий из следующих трех групп Java-классов:

- org.itbu.models — группа классов, отвечающих за хранения данных для компонентов Swing;
- org.itbu.models.jdbc — группа классов, отвечающих за доступ к базе данных, включая менеджер сессий и классы-обработчики данных для объектов из базы данных;
- org.itbu.models.pojo — классы, описывающие данные отношений (таблиц) базы данных.

Результат выполнения лабораторной работы № 4 представлен на рис. 2

Лабораторная работа № 5 "Разработка корпоративного клиент-серверного приложения на платформе Java EE 5". Целью этой работы является освоение технологии трехзвенного клиент-серверного приложения.

Трехзвенная архитектура является одним из шаблонов проектирования и используется многими архитекторами

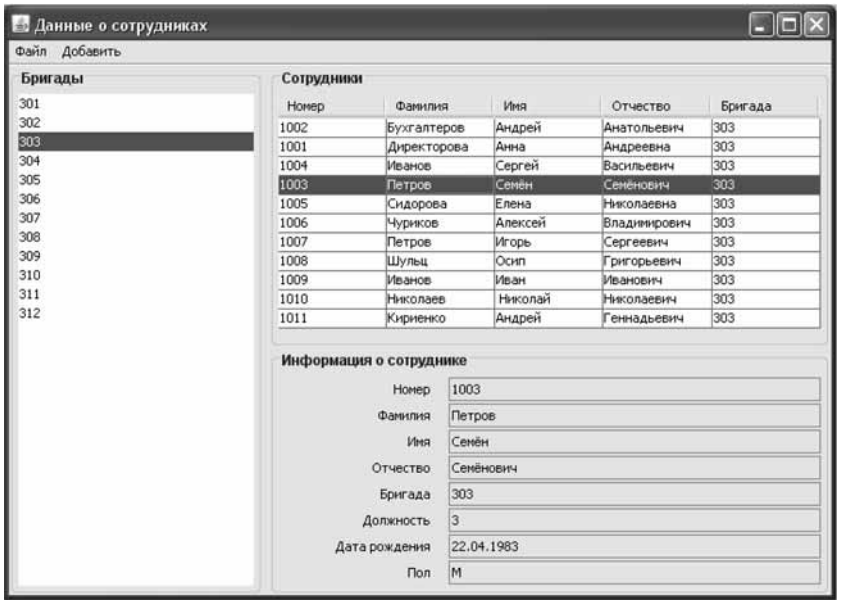


Рис. 2. Результат выполнения лабораторной работы № 4

при разработке приложений на основе спецификации Enterprise Java Beans (EJB) [3]. Эта спецификация входит в состав платформы Java EE, которая описывает компоненты, располагающиеся на стороне сервера и реализующие бизнес-логику корпоративного приложения. Рассматриваются спецификации и аннотации Java Persistence API с введением нового стиля доступа к данным, основанного на технологии POJO. Приложения взаимодействуют с Java-объектами во время выполнения, используя специальный объект, который называется entity manager. Архитектура корпоративного приложения рассматриваемого стиля программирования представлена на рис. 3.

Для выполнения практических заданий данной лабораторной работы в создаваемом приложении предлагается использовать базу данных, созданную в лабораторной работе № 1.

Студенту предлагается выполнить ряд заданий, связанных с созданием проектов EAR, JPA, EJB,

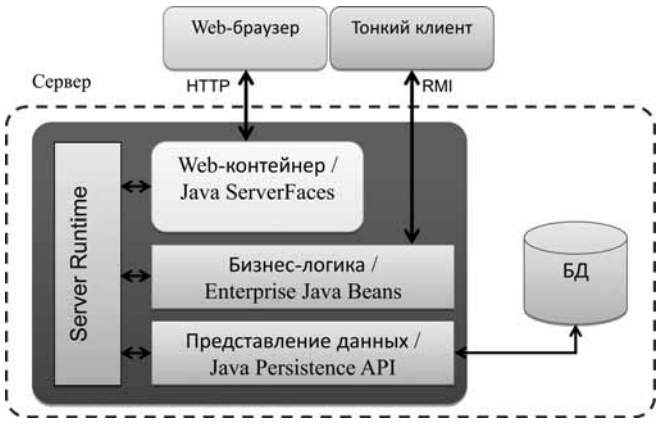


Рис. 3. Типовая клиент-серверная организация приложения

Application Client [3] для организации клиент-серверного решения функционирования приложения. В эти проекты вставляются необходимые программные Java-фрагменты, настраивается взаимодействие между ними, добавляются необходимые библиотеки и jar-файлы.

Настройка среды разработки Eclipse в данной лабораторной работе осуществляется самим студентом по отдельному заданию, в рамках которого выполняются следующие действия.

1. Определение драйвера СУБД DB2.
2. Установка соединения с базой данных в диалоговом окне Data Source Explorer.
3. Выбор и настройка связи среды Eclipse с сервером приложений WebSphere в окне Servers.
4. Работа с административной консолью сервера в целях настройки соединения базы данных с сервером приложений посредством организации более эффективного обращения к базе через пул данных (Database Pools).
5. Выполнение задания по развертыванию приложения на сервере WebSphere Application Server CE. Клиентское приложение также требует развертывания на сервере, как и остальные модули (проекты). Однако отличие в том, что выполняется оно в клиентской среде исполнения (на Java-машине клиента). На сервере лишь указывается, что у данного приложения существует клиент.
6. Указание с какими компонентами серверной среды будет взаимодействовать приложение, это необходимо в качестве настройки для развертывания на сервере EAR-проекта. После этого необходимо осуществление публикации этих модулей на сервере.

Все перечисленные выше действия достаточно подробно с пояснениями описываются так, чтобы студент без помощи преподавателя мог понять и выполнить лабораторную работу. Разбираются наиболее вероятные ошибки.

Таким образом, в интеграцию среды разработки для лабораторной работы № 5 добавляется еще и сервер WebSphere Application Server CE. Отмечается, что при переносе полученного и отлаженного приложения на другой компьютер, необходимо выполнить настройки, обозначенные выше в пунктах 1—6. Если эти настройки выполняются перед прохождением студентами практикума, среда разработки будет работоспособной для выполнения лабораторной работы на этом компьютере.

Результатом выполнения данной работы является выдача табличной формы списка сотрудников (рис. 4) средствами разработанного клиентского приложения, которое запускается с другого компьютера (клиентской рабочей станции) в виде файла с расширением .jar.

Лабораторная работа № 6 "Разработка корпоративного web-приложения на платформе Java EE 5". Целью

Рис. 4. Результат выполнения лабораторной работы № 5

данной работы является освоение web-технологии для реализации корпоративного приложения.

В данной лабораторной работе рассматриваются технологии JSP (JavaServer Pages) [3] и JSF [4] (JavaServer Fases) на основе EJB-контейнера, разработанного в лабораторной работе № 5. Лабораторная работа № 6 выполняется с использованием навыков и результатов, полученных в ходе выполнения лабораторной работы № 5. Технология JSP поясняется слушателям и реализуется в рамках широко известного шаблона Model-View-Controller (MVC, Модель—представление—поведение или Модель—представление—контроллер). Архитектура программного обеспечения в этом случае такова, что модель данных приложения, пользовательский интерфейс и управляющая логика разделены на три отдельных компонента:

- модель (Model). Модель предоставляет данные (обычно для View), а также реагирует на запросы (обычно от контроллера), изменяя свое состояние;
- представление (View). Отвечает за отображение информации (пользовательский интерфейс);
- поведение (Controller). Интерпретирует данные, введенные пользователем, и информирует модель и представление о необходимости соответствующей реакции.

Модификация одного из компонентов при этом оказывает минимальное воздействие на другие.

В рамках выполнения лабораторной работы № 6 рассматривается также широко известная технология JSF, которая подобно средствам Swing и AWT представляет собой каркас разработки приложений, предоставляющий набор стандартных графических компонентов для создания интерфейсов. Эта технология также как и JSP ориентирована на создание web-приложений. Сопоставляются возможности обоих технологий.

В практической части работы студентам предоставляется возможность: освоить работу с такими программными реализациями, как сервлеты, JSP-страницы, динамические web-проекты; использовать web-page-редактор, редактор навигации, аннотацию для связи с EJB-компонентом; вставлять Java-код в разметку JSP-страниц.

Студентам предлагают заготовки кодов, с которыми они знакомятся и разбираются. Заготовки модифицируют под отдельные постановки задач, отлаживают и сдают преподавателю в качестве зачетных единиц.

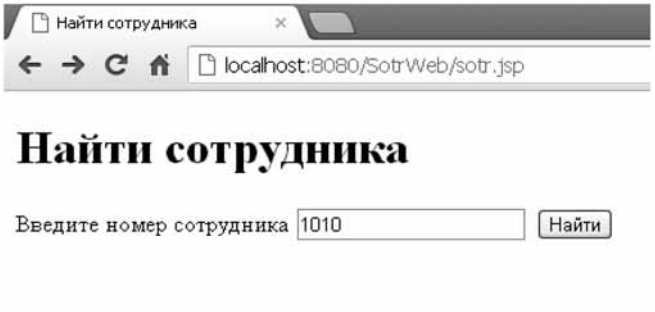


Рис. 5. Web-интерфейс выборки сотрудника по заданному идентификационному номеру

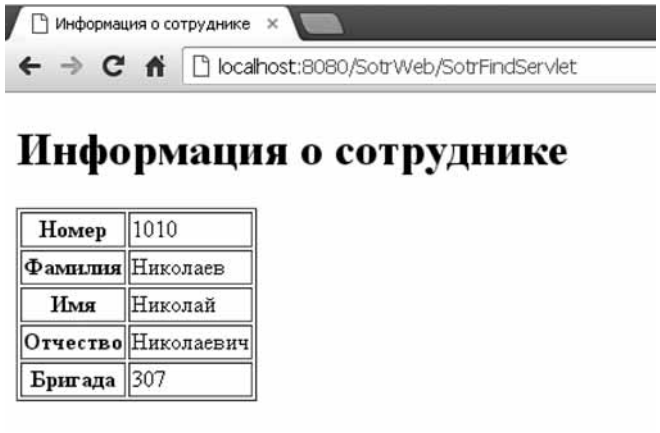


Рис. 6. Web-форма выдачи данных о найденном сотруднике

Примеры результатов выполнения лабораторной работы № 6 представлены на рис. 5—7.

В лабораторной работе № 6, как и в следующей, настройка интегрированной среды выражается в пополнении Eclipse-продукта дополнительными функциональными возможностями посредством подключения к нему других плагин-модулей. Такие модули позволяют выполнять задания с использованием JSF-технологии, а также технологии дизайнера отчетов Eclipse BIRT. Эти настройки выполняются системным администратором.

Лабораторная работа № 7 "Создание отчетов с помощью Eclipse BIRT". Целью этой работы являются освоение технологии работы в среде дизайнера отчетов и создание отчетов на основе источников данных jdbc и SQL-запросов.

Выполнение данной лабораторной работы начинается со знакомства с дизайнером отчетов Eclipse BIRT. Этот типовой редактор отчетов позволяет с помощью паллеты визуальных элементов формировать отчеты различной сложности обработки данных реляционного типа и различного вида. К таким видам относятся данные анкетного, табличного и других видов, с сортировкой, группировкой, фильтрацией и агрегацией результатов выдачи. Настройка программных средств среды разработки для этой лабораторной работы за-

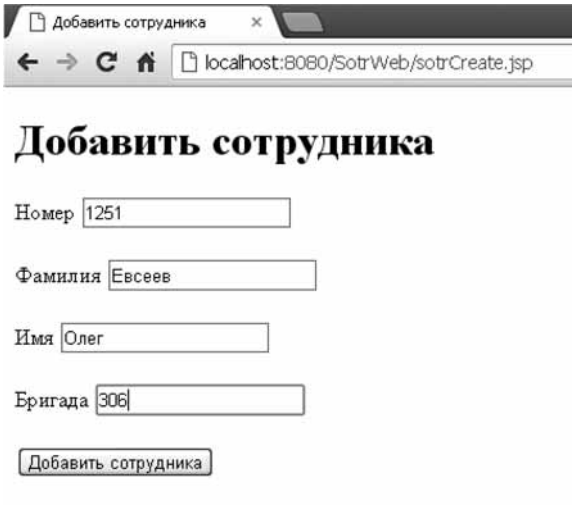


Рис. 7. Web-интерфейс ввода данных сотрудника, принятого на работу

Рис. 8. Рабочая область студента Иванова, создаваемая им в результате выполнения всего практикума

ключается в подключении плагинов, добавляющих функции генерации отчетов.

Таким образом, конфигурирование интегрированной среды выполнения лабораторных работ по созданию программного обеспечения информационных систем можно разделить на следующие этапы.

1. Установка трех основных IBM-продуктов и компонентов на индивидуальный компьютер.

2. Выполнение системным администратором на данном компьютере настроек (назовем их системными настройками). К ним относятся: плагин-настройки, установка Apache HTTP-сервера, браузера FireFox, виртуальной машины Java, библиотеки Jdk1.6.22, Java Runtime Environment (JRE6), визуального конструктора (Visual Editor) и других элементов.

3. Выполнение заданий лабораторных работ, связанных с дополнительными настройками силами самих студентов среды разработки и функционирования полученных в ходе разработки результатов для освоения ими этих способов настройки (назовем их студенческими настройками).

При установке работоспособной среды практикума системный администратор должен выполнить первые два пункта конфигурирования средств интеграции на каждом из используемых в работе компьютеров. Для повышения эффективности установки интегрированной среды выполнения практикума планируется создать ее зеркальную копию с учетом системных настроек, упомянутых в п. 2 перечисленных выше этапов.

В целях проверки эффективности инструментальной среды выполнения практикума, в одной рабочей области с условной фамилией студента создана тестовая версия выполняемых работ и в ней отлажены проекты по всем лабораторным работам. Проводник проектов показан на рис. 8.

Процесс проверки работоспособности и эффективности среды выполнения практикума заключается в том, чтобы "прогнать" тестовую версию лабораторного комплекса с учетом студенческих настроек на всех персональных компьютерах, которые задействованы в практикуме. При этом на каждом компьютере устанавливается копия тестовой рабочей области и запускается каждая лабораторная работа. Отметим, что в предварительном порядке выполняются студенческие настройки. Легче повторить настройки, чем повторить для проверки полное выполнение лабораторных работ. Если после таких действий компьютер выполняет все возложенные на него функции, то можно считать, что он подготовлен для использования в лабораторном практикуме.

* * *

По опыту внедрения рассмотренного лабораторного практикума в учебный процесс можно сделать некоторые выводы:

- студенты проявляют интерес к освоению практикума, который предоставляет им хорошие возможности получить реальные навыки работы с продуктами IBM, что позволяет им создавать поэлементно информационные системы достаточно сложной архитектуры, настраивая средства разработки;
- студентам можно усложнять задачи, наращивая возможности среды Eclipse путем подключения разных плагинов и даже предлагать другие продукты компании IBM, новые Java технологии и модели разработки информационных систем.

Список литературы

1. **Современные** технологии создания программного обеспечения. Обзор. URL: http://citforum.ru/programming/application/program/lit.shtml

2. **Введение** в интегрированную среду разработки Eclipse. http://www.javaportal.ru/java/ide/intro_eclipse.html

3. **Фанг Д.** и др. Введение в IBM RationalApplication Developer. Уч. руководство (в комплекте CD) / Пер. с англ. М.: КУДИЦ-ПРЕСС, 2006. 592 с.

CONTENTS

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 04.12.2012 г. Подписано в печать 23.01.2012 г. Формат 60×88 1/8. Заказ PI213
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
105120, г. Москва, ул. Нижняя Сыромятническая, д. 5/7, стр. 2, офис 2.