Российская академия наук
Сибирское отделение
Институт систем информатики
имени А. П. Ершова

Елена Викторовна Касьянова

АДАПТИВНЫЕ МЕТОДЫ И СРЕДСТВА ПОДДЕРЖКИ ДИСТАНЦИОННОГО ОБУЧЕНИЯ ПРОГРАММИРОВАНИЮ

Под редакцией
проф. Виктора Николаевича Касьянова

Новосибирск 2007

УДК 519.68; 681.3.06
ББК З 22.183.49+З 22.174.2

Книга посвящена разработке адаптивных методов и средств поддержки дистанционного обучения программированию в рамках проблемного подхода и является четырнадцатой в серии, издаваемой Институтом систем информатики имени А.П. Ершова СО РАН по проблемам конструирования и оптимизации программ. Рассмотрены вопросы дистанционного и проблемного обучения. Исследованы методы и средства адаптации и интеллектуализации дистанционного обучения. Описана адаптивная среда WAPE поддержки дистанционного обучения программированию в рамках проблемного подхода. Представлен вводный курс программирования на базе нового языка Zonnon, недавно пополнившего известное семейство Цюрихских языков: Паскаль, Модула-2 и Оберон.

Книга представляет интерес для системных программистов, а также студентов и аспирантов, специализирующихся в области системного и теоретического программирования. Будет особенно интересна тем, кто связан с разработкой методических и программных средств поддержки дистанционного обучения.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

E.V. Kasyanova

ADAPTIVE METHODS AND TOOLS FOR SUPPORT OF DISTANCE EDUCATION IN PROGRAMMING

Edited by
prof. V. N. Kasyanov

Novosibirsk 2007

The book is devoted to the development of adaptive methods and tools for support of distance education in programming in the framework of the problem-based approach. It is the fourteenth volume in a series of books published by A.P. Ershov Institute of Informatics Systems SB RAS on problems of program construction and optimization. The problems of distance and problem-based education are considered, as well as the methods and tools of its adaptation and intellectualization. The adaptive environment WAPE for support of distance education in programming in the framework of the problem-based approach is described. An introductory course of programming on the base of a new language Zonnon is presented.

The book is intended for system programmers, students and post-graduates working in the field of system and theoretical programming. It will be of particular interest for those connected with the development of methodological and program tools for support of distance education.

ВВЕДЕНИЕ

В не столь далеком прошлом хороший почерк был гарантией спокойной и обеспеченной жизни до старости. Последние десятилетия характерны ускорением обновляемости технологий и знаний в различных сферах деятельности человека. Поэтому школьного и даже вузовского образования надолго уже не хватает. Сегодня особенно актуальна концепция непрерывного образования на протяжении всей жизни или, как говорят, пожизненного обучения (long-life education). Поиск соответствующей организационной структуры и учреждений образования (особенно образования взрослых), которые обеспечили бы переход от принципа «образование на всю жизнь» к принципу «образование через всю жизнь» — важнейшая проблема XXI века. Открытое образование — это образование, доступное всем. Его развитие неизбежно приведет к существенному пересмотру традиционных методик и технологий учебного процесса, а также к формированию единого открытого образовательного пространства на основе дистанционного обучения.

Системы дистанционного обучения в настоящее время активно исследуются и развиваются и уже успели пройти путь в пять поколений, начиная от систем обучения по переписке, больше известных в СССР как системы заочного обучения, и кончая системами гибкого обучения и интеллектуального гибкого обучения, определяющими настоящее и будущее дистанционного образования и базирующимися на Web-технологиях. Выгоды сетевого обучения ясны: аудиторная и платформенная независимости. Сетевое обучающее программное обеспечение, один раз установленное и обслуживаемое в одном месте, может использоваться в любое время и по всему миру тысячами учащихся, имеющих компьютеры, подключенные к Интернету. Тысячи программ сетевого обучения и других образовательных приложений стали доступны в сети за последние годы. Проблема состоит в том, что большинство из них является не более чем статичными гипертекстовыми страницами и не поддерживает проблемный подход к обучению.

Вместе с тем учебный процесс представляет собой совместную деятельность обучающего и обучаемого, который нельзя осуществить без решения задач, хотя в отличие от других видов деятельности (например, производственной или познавательной), где результаты решения задач являются ее прямыми продуктами, в учебной деятельности решение задач — это не цель, но средство достижения целей, а именно, учебных целей, направленных на изменение обучаемого, а не предметов внешнего мира. Особенно важен проблемный подход при начальном обучении программированию, в

процессе которого обучаемый должен овладеть навыками точного формулирования алгоритмов на языке высокого уровня. Это невозможно сделать, прочитав несколько руководств или прослушав курс лекций по программированию. Необходима практика конструирования алгоритмов, и здесь невозможно обойтись без подходящего набора примеров и задач, а также без оценки разработанных алгоритмов на правильность и качество.

Появившиеся в последнее время адаптивные гипермедиа-системы существенно повышают возможности обучающих систем. Целью адаптивных систем является персонализация гипермедиа-системы, ее настройка на особенности индивидуальных пользователей. Поддержка адаптивных методов в гипермедиа-системах оказывается весьма полезной в тех случаях, когда имеется одна система, обслуживающая множество пользователей с различными целями, уровнем знаний и опытом, и когда лежащее в ее основе гиперпространство является относительно большим. Поэтому области применения адаптивной гипермедиа выходят далеко за границы обучающих систем и включают, например, такие, казалось бы, далекие от обучения области применения гипермедиа-систем, как открытые адаптивные виртуальные музеи.

Обучающие гипермедиа-системы, в которых пользователь или ученик имеет конкретную цель обучения (включая и такую цель, как общее образование), являются типичным приложением адаптивных гипермедиа-систем. В этих системах основное внимание уделяется знаниям обучающихся, которые могут сильно различаться. Состояние знаний изменяется во время работы с системой. Таким образом, корректное моделирование изменяющегося уровня знаний, надлежащее обновление модели и способность делать правильные заключения на базе обновленной оценки знаний являются важнейшей составляющей обучающей гипермедиа-системы.

Данная книга посвящена вопросам разработки адаптивных методов и средств поддержки дистанционного обучения программированию в рамках проблемного подхода. Исследования, представленные в книге, выполнялись в соответствии с планами научно-исследовательских работ ИСИ СО РАН по проекту 3.15 «Методы и средства трансляции и конструирования программ» программы 3.1 СО РАН «Информационное и математическое моделирование в различных областях знаний, задачи поддержки принятия решений, экспертные системы, системное и теоретическое программирование» и поддерживались грантами Минобразования (грант УР.04.01.023 «Графы в программировании: обработка, визуализация и применение» научной программы «Университеты России» и грант Е02-1.0-42 «Граф-модели в программировании и визуальная обработка» по фундаментальным

исследованиям в области естественных и точных наук) и Microsoft Research (грант «Разработка вводного курса программирования»).

Книга состоит из четырех глав и приложения.

Первая глава посвящена вопросам дистанционного и проблемного обучения. Глава начинается с рассмотрения проблем российского образовательного рынка и целей государственной программы по созданию систем открытого образования на основе технологий дистанционного обучения (разд 1.2). Описываются основные этапы в развитии дистанционного обучения, связанные с использованием разных моделей (модель обучения по переписке, мультимедийная модель обучения, модель телеобучения, модель гибкого обучения и модель интеллектуального гибкого обучения) и технологий (разд. 1.3). Исследуются особенности учебной деятельности и проблемного подхода к обучению (разд.1.3). Завершают изложение выводы по главе.

Вторая глава посвящена вопросам обеспечения персонализации при дистанционном обучении, особое внимание в ней уделяется анализу методов и средств адаптивной гипермедиа, используемых современными адаптивными обучающими Web-системами.

В разд. 2.1 определяются адаптивные гипермедиа-системы и дается ответ на вопрос, к чему они адаптируются, а также рассматриваются характеристики пользователя обучающей системы, важные для ее адаптации. Вопросу, что адаптируется в системах Web-обучения, посвящен разд. 2.2. В разд. 2.3 рассматриваются такие технологии адаптации, как выстраивание программы обучения, интеллектуальный анализ решений обучаемого, поддержка в решении задач, привнесенные в обучающие Web-системы из несетевых интеллектуальных обучающих систем. Разд. 2.4 и 2.5 содержат описание методов и технологий адаптации содержания и навигации, характерных для адаптивных гипермедиа-систем. Моделям предметной области, пользователя и адаптации, используемым в адаптивных гипермедиа-системах, посвящен разд. 2.6. Разд. 2.7 содержит примеры существующих экспериментальных адаптивных обучающих Web-систем. Завершают изложение выводы по главе 2.

Третья глава посвящена проекту WAPE, работа над которым ведется в Институте систем информатики СО РАН. Цель проекта — разработка адаптивной среды дистанционного обучения WAPE, поддерживающей активное индивидуальное обучение программированию в рамках проблемного подхода и соединяющей возможности адаптивных гипермедиа-систем и интеллектуальных обучающих систем.

Глава начинается с общего описания системы WAPE и ее возможностей с точки зрения обучаемых (студентов). Возможностям, которые система предоставляет другим пользователям (администраторам, лекторам и инструкторам), посвящен разд. 3.2. В разд. 3.3 рассматривается модель знания курса, а в разд. 3.4 и 3.5 — вопросы моделирования знаний студента и использования при этом моделировании механизма сетей Байеса. Моделям тестирования и видам тестов посвящен разд. 3.6. В разд. 3.7 и 3.8 рассматриваются проекты (упражнения и задания) подсистемы CLASS, играющей роль виртуальной семинарской аудитории в рамках системы WAPE. Разд. 3.9 содержит описание подсистемы PRACTICE, предназначенной для использования при прохождении студентами компьютерного практикума по курсу, поддерживаемому системой WAPE. Виды аннотирования проектов и другие действия, которые осуществляет система в помощь студентам, рассматриваются в разд. 3.10. Разд. 3.11 посвящен режимам работы студентов, а вопросы обновления модели знаний студента и развития курса описаны в разд. 3.12 и 3.13. В раз. 3.14 описываются работы по экспериментальной реализации системы WAPE. Завершают изложение выводы по главе 3.

Четвертая глава посвящена вводному курсу программирования на базе языка Zonnon.

При обучении программированию наиболее важным представляется начальный этап, на котором обучаемый должен овладеть навыками точного формулирования алгоритмов на языках высокого уровня. Это невозможно сделать, прочитав несколько руководств или прослушав курс лекций по программированию. Необходима практика конструирования алгоритмов, и здесь невозможно обойтись без языка программирования, пригодного для целей начального обучения, и подходящего набора примеров и задач.

В главе кратко представлен вводный курс программирования на базе языка Zonnon, описаны две книги: «Введение в программирование» и «Практикум по программированию», поддерживающие этот курс.

Zonnon — это новый универсальный язык программирования в семействе языков Паскаль, Модула-2 и Оберон, работа над которым ведется в Цюрихском институте информатики. Он сохраняет стремление к простоте, ясному синтаксису и независимости концепций, а также уделяет внимание параллельности и легкости композиции и выражения. Унификация абстракций является стержнем проектирования языка Zonnon, и она отражается в его концептуальной модели, основанной на модулях, объектах, определениях и реализациях. Язык Zonnon содержит такие новые черты, как активность в объектах, основанный на межобъектном взаимодействии диалог, перегрузка операций и обработка исключительных ситуаций. Язык Zonnon

специально разрабатывается как платформно-независимый язык. Первая реализация языка Zonnon выполнена для платформы .NET. Кроме того, предполагается интеграция компилятора в систему программирования Visual Studio .NET (в сотрудничестве с компанией Microsoft).

Глава начинается с изложения основных особенностей языка Zonnon (разд. 4.1). Разд. 4.2 содержит описание цели и основных принципов курса. Разд. 4.3 является кратким описанием книги «Введение в программирование», являющейся учебником по курсу. Цели и содержание компьютерного практикума составляют разд. 4.4. Разд. 4.5 содержит описание книги «Практикум по программированию», предназначенной для поддержки практикума. Вопросы использования курса в рамках системы WAPE рассматриваются в разд. 4.6. Главу завершают выводы по главе.

Приложение содержит полное описание языка Zonnon в его текущей версии. Компилятор и среду программирования для данной версии языка можно скачать с сайта разработчиков языка Zonnon [156]. Там же всегда можно найти последние новости о языке, его реализации и применении, а также обратиться за консультацией к разработчикам.

ГЛАВА 1. ДИСТАНЦИОННОЕ ОБУЧЕНИЕ И ПРОБЛЕМНЫЙ ПОДХОД

Последние десятилетия характерны ускорением обновляемости технологий и знаний в различных сферах деятельности человека. Поэтому школьного и даже вузовского образования надолго уже не хватает. Сегодня особенно актуальна концепция непрерывного образования на протяжении всей жизни или, как говорят, пожизненного обучения (long-life education). Поиск соответствующей организационной структуры и учреждений образования (особенно образования взрослых), которые обеспечили бы переход от принципа «образование на всю жизнь» к принципу «образование через всю жизнь» — важнейшая проблема XXI века. Открытое образование — это образование, доступное всем. Его развитие неизбежно приводит к существенному пересмотру традиционных методик и технологий учебного процесса, а также к формированию единого открытого образовательного пространства на основе дистанционного обучения.

Данная глава посвящена вопросам дистанционного и проблемного обучения. Глава начинается с рассмотрения проблем российского образовательного рынка и целей государственной программы по созданию систем открытого образования на основе технологий дистанционного обучения (разд. 1.1). В разд. 1.2 описываются свойства пяти поколений систем дистанционного обучения, начиная от систем обучения по переписке, больше известных в СССР как системы заочного обучения, и кончая системами гибкого обучения и системами интеллектуального гибкого обучения, определяющими настоящее и будущее дистанционного образования и базирующимися на Web-технологиях. В разд. 1.3 исследуются особенности учебной деятельности и проблемного подхода к обучению. Завершают главу выводы.

1.1. Дистанционное обучение и открытое образование

Систематические исследования в области компьютерной поддержки профессионального образования имеют более чем 30-летнюю историю. За этот период в учебных заведениях США, Франции, Японии, России и ряда других стран было разработано большое количество компьютерных систем учебного назначения, ориентированных на различные типы ЭВМ. Однако сферы применения таких систем гораздо шире. Это крупные промышленные предприятия, военные и гражданские организации, ведущие самостоятельную подготовку и переподготовку кадров. Кроме того, в развитых странах становится уже стандартом снабжать новые сложные машины и техно-

логии компьютерными обучающими системами, облегчающими и ускоряющими процесс их освоения и внедрения. За рубежом разработку «мягкого» компьютерного продукта учебного назначения (методических и программно-информационных средств) считают весьма дорогостоящим делом в силу его высокой наукоемкости и необходимости совместной работы высококвалифицированных специалистов: психологов, преподавателей-предметников, компьютерных дизайнеров, программистов. Несмотря на это, многие крупные зарубежные фирмы финансируют проекты создания компьютерных учебных систем в учебных заведениях и ведут собственные разработки в этой области.

Появление и активное распространение дистанционных форм обучения на основе сети Интернет является адекватным откликом систем образования многих стран на происходящие в мире процессы интеграции, движения к информационному обществу. В Европе и Северной Америке создаются консорциумы ведущих университетов, предоставляющих широкий спектр дистанционных образовательных услуг. Так, ассоциация дистанционного обучения в США объединяет в своем составе пять тысяч учебных заведений. Юнеско ведет работу по организации распределенного университета, обучение в котором будет происходить в виртуальном пространстве, вне зависимости от расселения и границ, без ограничений по времени. Обучение на расстоянии позволяет получить «столичное» и даже международное образование жителям провинции, военнослужащим, детям-инвалидам и другим категориям граждан. К тому же стоимость таких образовательных услуг гораздо ниже, чем при традиционном обучении.

Несмотря на трудные экономические условия, Россия не может и не остается в стороне от развития этих процессов.

На конец XX-го века образовательный рынок в России характеризуется следующими данными: не менее 36,5 млн. российских граждан в возрасте от 18 до 45 лет (более 1/3 взрослого населения) имеют актуализированные образовательные потребности, из них более 1/2 хотели бы получить профессиональное образование, около 1/2 нуждаются в дополнительном обучении; более 70% были бы готовы в той или иной мере оплачивать свое обучение. Существующая в России система профессионального образования даже теоретически не способна удовлетворять образовательные потребности в таких объемах.

Особенностью России является и проблема доступности. Исследования иркутских социологов: 50% желающих не могут поступить в вузы. При этом в развитых странах студенты составляют от 3 до 6% всего населения, а это от 80 до 89% выпускников школ, что существенно превышает россий-

ский показатель в 15,6% числа студентов от числа выпускников. В России должно быть около 7 млн. студентов. Причины: противоречие между возрастающим спросом на образование и возможностями их реализации; вступительные экзамены в вузы как наиболее коррупционный этап учебного процесса.

Имеет место неудовлетворенность населения России и качеством образования. Куда предпочли бы поступить абитуриенты российских государственных вузов? Анализ социологов: 50% — престижный московский вуз; 23,5% — вуз в своем городе; 16,3% — зарубежный университет; 4,3% — престижный вуз в Санкт-Петербурге; 5,9% — затруднились ответить. Вывод социологов: более 70% абитуриентов поступают не в тот вуз, в который хотели бы.

Есть и другая сторона проблемы. Исследования НИИ психологии обучения и социологии образования России: студенты московских вузов прогуливают в среднем каждое третье занятие; 45% опрошенных московских студентов объясняют, что им неинтересно учиться. Исследования Новосибирского кадрового агентства «Офис-центр»: 58% выпускников вузов не желают работать по специальности (Прим.: не по специальности работают до 30% выпускников МГУ, до 80 % выпускников сельскохозяйственных вузов); 80% студентов-выпускников к моменту окончания вуза уже имеют опыт работы, из них 2/3 подрабатывают не по специальности. Отсюда и проблемы занятости.

Исследования Института проблем занятости РАН: доля студентов дневных отделений, намеренных продолжить свое образование в форме вечернего обучения: в 1998 г. — 22,5%, а в 1999 г. — уже 34,3%; доля студентов дневных отделений, намеренных продолжить свое образование в форме заочного обучения: в 1998 г. — 6,3%, а в 1999 г. — уже 10,4 %. С другой стороны, выпускники высших учебных заведений ощущают проблемы трудоустройства: в 1998 году — 30,6%, а в 1999 г. — уже 42,2%. И выпускники на перепутье: ¼ выпускников всех видов учебных заведений ежегодно обращаются в московскую службу занятости, из них 60% признаются сразу безработными, а только 20% — трудоустраиваются. Доля лиц, имеющих высшее и среднее профессиональное образование, в общей численности официальных безработных: в 1998 г. — 32,4%, а в 1999 г. — уже 35,4%. При этом дипломированные специалисты в Японии составляют 40% населения, в Финляндии — около 20%, в России — только 7,6%. В технократических странах (Япония, США и др.), ставящих задачу всеобщего высшего образования, тратится на это 15–20% государственного бюджета.

Поэтому не случайно в конце 2000 г. Правительством России была принята государственная программа по созданию системы открытого образования (ОО). Цель программы — обеспечение общенационального доступа к образовательным ресурсам путем широкого использования информационных образовательных технологий дистанционного обучения и на этой основе предоставление условий для наиболее полной реализации гражданами своих прав на образование, по структуре и качеству соответствующих потребностям развития экономики и гражданского общества.

Обеспечение возможности для получения высшего образования и обучения на протяжении всей жизни, предоставление учащимся права свободного выбора места, времени и технологий обучения в рамках системы ОО, наряду с индивидуальным развитием и социальной мобильностью, позволяют сохранять, развивать и распространять национальные и региональные, международные и исторические культуры в условиях культурного плюрализма, содействовать воспитанию молодежи в духе ценностей, составляющих основу демократической гражданственности. Открытость образования предполагает: открытое поступление в высшее учебное заведение (как правило, без анализа исходного уровня знаний, без вступительных испытаний: политика «открытых дверей»); открытое планирование обучения (свобода составления индивидуальной образовательной траектории — модулей из системы учебных курсов соответствующей программы); свободу выбора преподавателя (определение того преподавателя, который в наибольшей степени потенциально соответствует потребностям, особенно в дальнейшем, когда обучение может перейти в образовательный консалтинг); свободу в выборе времени, ритма и темпа обучения (прием на обучение в течение всего года, отсутствие фиксированных сроков обучения); свободу в выборе места обучения (самостоятельный выбор территории обучения).

Система ОО ориентирована на массовость и общедоступность независимо от социального статуса, территориального расположения, ограничения в гражданских правах и т.п.; на обеспечение широкого доступа к национальным и мировым образовательным ресурсам; на возможность получения второго образования, например, экономического образования специалистами естественнонаучного профиля, технического образования специалистами медицинского профиля и т.п. Эта система должна стать таким социальным институтом, который был бы способен предоставить человеку разнообразные образовательные услуги, позволяющие учиться непрерывно, и обеспечить возможность получения современного профессионального знания. Подобная система дает возможность каждому обучаемому выстроить ту образовательную траекторию, которая наиболее полно соответствует

его образовательным и профессиональным способностям, где бы территориально он ни находился. В итоге должна быть сформирована ассоциация (консорциум) связанных друг с другом учебных учреждений, которая обеспечивает создание пространства образовательных услуг, взаимосвязь и преемственность программ, способных удовлетворять запросы и потребности населения. Таким образом, создается возможность многомерного движения специалиста в образовательно-профессиональном пространстве, развития через обучение и функционирование постоянного образовательного профессионального консалтинга.

Текущее состояние создаваемой системы ОО пока еще далеко от указанных радужных перспектив. Возможности современных учебных учреждений дистанционного обучения можно проследить на примере ИнтУИТ-университета — первого российского «Интернет-университета информационных технологий», который открылся в 2003 г. и предлагает всем желающим совершенно бесплатно получить дополнительное (послевузовское) профессиональное IT-образование, соответствующее мировым стандартам [14]. По результатам обучения университет выдает либо удостоверение, когда все обучение велось дистанционно, либо диплом, когда обучаемый приезжал для очной сдачи экзаменов. Учебно-методический центр ИнтУИТ-университета ведет разработку и внедрение собственных учебных курсов по дисциплинам, связанным с информационными технологиями. В рамках проекта планируется создать более 100 различных курсов и выпустить по ним учебники. Для сравнения отметим, что Международное общество IEEE (Institute of Electrical and Electronic Engineers) [40] в настоящее время предлагает 1300 дистанционных курсов по данной тематике. К настоящему времени университетом разработана и поддерживается почти половина из запланированных курсов, учебники по которым опубликованы на сайте и доступны в печатном виде, а также на CD. Процесс обучения в ИнтУИТ-университете аналогичен процессу заочного обучения (обучения по переписке) с тем отличием, что для доставки учебников, заданий и решений используются не только обычная почтовая служба, но и клиент-серверные технологии и электронная почта.

1.2. Пять поколений систем дистанционного обучения

Многие годы университеты, активно работающие в области дистанционного и открытого образования, находились и находятся на передовых позициях в использовании новых технологий для увеличения доступа к возможностям обучения и подготовки. Модели дистанционного обучения,

используемые ими, постепенно эволюционировали, пройдя через следующие четыре стадии [143]: модель обучения по переписке, мультимедийная модель обучения, модель телеобучения, модель гибкого обучения.

Модель обучения по переписке поддерживается системами дистанционного обучения первого поколения, которые основываются на печатной технологии и являются гибкими с точки зрения времени, места и пространства. Эти системы не обладают высокой интерактивностью, но предоставляют хорошо проработанный учебный материал.

Мультимедийная модель характеризуется системами дистанционного обучения второго поколения, которые, сохраняя гибкость с точки зрения времени, места и пространства и хорошую проработанность учебного материала, используют в качестве технологий доставки не только печатную продукцию, но и аудиокассеты и видеозаписи, а также такие высоко интерактивные средства, как компьютерное обучение и интерактивное видео.

Модель телеобучения связана с системами дистанционного обучения третьего поколения, базирующимися на следующих технологиях: аудиоконференция, видеоконференция, широковещательное ТВ/радио и аудиовидеоконференции. Эти системы привязаны к определенному времени, месту и пространству, обладают высокой интерактивностью, но в случае с аудиовидеоконференциями могут поставлять не вполне проработанный учебный материал.

Модель гибкого обучения поддерживается системами дистанционного обучения четвертого поколения, использующими в качестве технологий доставки интерактивный мультимедийный диалог, основанный на Интернете, доступ к WWW-ресурсам и компьютерное взаимодействие. Эти системы обладают высокой интерактивностью и хорошо проработанным учебным материалом.

И хотя в последние годы многие университеты только начинают разворачивать системы дистанционного обучения четвертого поколения, эволюционный процесс продолжается, и на основе дальнейшего использования новых технологий уже начинают появляться системы дистанционного обучения пятого поколения, поддерживающие так называемую *модель интеллектуального гибкого обучения* [143].

Системы дистанционного обучения пятого поколения используют для доставки следующие технологии: интерактивный мультимедийный диалог, основанный на Интернете; доступ к WWW ресурсам; компьютерное взаимодействие на основе автоматизированных ответных систем; доступ через портал кампуса к процессам и ресурсам учреждения. Эти системы высоко интерактивны и предоставляют хорошо проработанный учебный материал.

Следует отметить, что системы первых трех поколений, а также системы четвертого поколения, опирающиеся на компьютерное взаимодействие, характеризуются переменной стоимостью, имеющей тенденцию к увеличению или уменьшению, напрямую связанной (часто линейно) с изменением объема активности; например, в системах доставки второго поколения распространение пакетов материалов для самостоятельного обучения имеет переменную стоимость, изменяющуюся прямо пропорционально числу обучаемых студентов. В отличие от таких систем, системы дистанционного обучения пятого поколения обладают тенденцией к значительному уменьшению стоимостей, связанных с обеспечением доступа к учережденческим процессам и онлайновому обучению отдельных пользователей.

Центральной составляющей систем пятого поколения является разработка принимаемого е-интерфейса, портала кампуса, через который студенты, преподаватели и другие штатные сотрудники могут связываться с университетом высокоинтерактивным и вынужденным образом. Для успеха на появляющемся глобальном рынке пожизненного обучения университету нужно создать портал кампуса, который достигнет такой степени интерактивности, пользовательской дружественности и адаптивности, которых сегодня нет у громадного большинства Web-сайтов кампусов.

1.3. Проблемный подход в обучении

Возникнув около сорока лет тому назад, деятельностный подход в обучении, опирающийся на работы Л.С. Выготского, А.Н. Леонтьева, С.Л. Рубинштейна и развитый в трудах Б.Ц. Бадмаева, П.Я. Гальперина, В.В. Давыдова, Е.И. Машбица, З.А. Решетовой, Н.Ф. Талызиной, Л.М. Фридмана, Г.П. Щедровицкого, Д.Б. Эльконина и др., превратился в законченную теорию учения, признанную в мире [1–3, 5, 9, 41, 42, 44–46, 50, 56].

Деятельностный подход в психологии вообще основан на принципиальном положении о том, что психика человека неразрывно связана с его деятельностью и ею обусловлена. При этом деятельность понимается как преднамеренная активность человека, проявляемая в процессе его взаимодействия с окружающим миром, и это взаимодействие заключается в решении жизненно важных для человека задач. Всякие действия и поступки человека определяются какими-либо потребностями. Под потребностью понимают состояние человека, отражающее его нужду в чем-либо, необходимом для его существования и развития. Такое состояние выступает в качестве источника активности человека. Человек не рождается с готовыми взгляда-

ми на мир, знаниями о нем, умением решать задачи. Осуществлять деятельность человеку позволяет усваиваемый им опыт общественно-исторической практики, опыт человечества, который передается с помощью старшего поколения. Этот опыт воплощен в предметах материальной и духовной культуры (орудиях и средствах производства, произведениях искусства, всевозможных носителях информации) и в способах выполнения действий с ними. Усвоение опыта общественно-исторической практики играет решающую роль в жизни человека на протяжении всей его жизни. При этом существуют два специально организуемых вида деятельности людей, в процессе которых они усваивают опыт предыдущих поколений — воспитание и учение. Нас в дальнейшем будет интересовать учение.

Деятельностный подход должен реализоваться и в учении, т.е. процесс учения необходимо рассматривать как деятельность. Для преподавателя это означает, что в процессе обучения, при передаче опыта общественно исторической практики он должен решать задачу формирования у обучаемых умения осуществлять деятельность. Следовательно, целью обучения также является деятельность, или действия и операции, с помощью которых она реализуется и которые направлены на решение специфических для учения задач. Систему операций, которая обеспечивает решение задач определенного типа, называют способом действий. Таким образом, конечной целью обучения является формирование способа действий.

Подход к процессу учения как к деятельности потребовал пересмотра взглядов на знания и умения, их роль и соотношение. Две традиционные задачи педагогики, заключающиеся в передаче знаний и в формировании умений по их применению и решающиеся последовательно, заменяются одной задачей. Знания и умения, или действия обучаемого, в которых эти умения реализуются, рассматриваются теперь не в противопоставлении друг другу, а в единстве. Это обусловлено тем, что усвоение знаний происходит одновременно с освоением способов действия с ними. Всякое обучение основам наук в то же время является и обучением соответствующим умственным действиям, а формирование умственного действия невозможно без усвоения определенных знаний. При этом первичными с точки зрения целей обучения являются действия, и это потребовало пересмотра содержания обучения. Его должны составлять не заданная система знаний (идеи, теории, другая научная информация) и затем усвоение этих знаний, как до сих пор считает педагогика, а заданная система действий и знания, обеспечивающие освоение этой системы. Знать — значит не просто помнить определенные знания (вдумайтесь, как часто отождествляются эти глаголы — знать и помнить), а выполнять определенную деятельность, связанную с

этими знаниями. Таким образом, знания становятся не целью обучения, а его средством. Они усваиваются для того, чтобы с их помощью выполнять действия, действовать, осуществлять деятельность, а не для того, чтобы они просто запоминались и служили только лишь повышению эрудиции.

Изложенные соображения являются основополагающими при проектировании обучения, которое должно начинаться с психологического анализа деятельности будущих специалистов. Только после этого могут быть определены необходимые знания, которые по отношению к деятельности играют служебную роль, они объясняют практические действия.

Центральным с точки зрения организации и функционирования системы образования является понятие учебного процесса. *Учебный процесс* представляет собой совместную деятельность обучающего и обучаемого. Деятельность обучающего в учебном процессе называют *обучением*, а деятельность обучаемого — *учением* или *учебной деятельностью*. Под учением понимают специально организованную деятельность людей, направленную на усвоение опыта предыдущих поколений. Применительно к высшей школе это означает, что обучаемый должен освоить способы действий, на которых основывается его будущая профессиональная деятельность.

Учебная деятельность является весьма специфическим видом деятельности и имеет ряд особенностей. Попутно заметим, что она представляет собой и цель, и продукт обучения. Любая деятельность начинается с потребности. Однако говорить о том, что, например, первоклассники идут в школу по потребности, наверное, не следует. У многих обучаемых нет потребности учиться, зато у их родителей есть большое желание дать своим детям образование. Таким образом, зачастую потребность в учебной деятельности опосредована.

Одной из основных особенностей учебной деятельности, отличающей ее от других видов деятельности, является то, что обучаемый — это не только субъект деятельности, но, одновременно, и ее объект. Это объясняется тем, что целью учебной деятельности являются изменения самого субъекта деятельности, а не преобразование объектов внешнего мира, не изменение предметов, с которыми взаимодействует субъект. Изменения в субъекте означают усвоение им определенных знаний, освоение им способов действий, соответствующих этим знаниям. Поскольку именно ради этого и организуется процесс обучения, усвоение знаний и освоение способов действий, совершение соответствующих им изменений субъекта являются прямым продуктом учебной деятельности. Таким образом, одна из особенностей учебной деятельности заключается в неотъемлемости ее продукта от ее субъекта. В любых других видах деятельности ее продукты отчуждаются

от ее субъектов, они служат другим людям. Продукты деятельности продаются, демонстрируются в картинных галереях, театрах, кино. Это относится и к познавательной (научно-познавательной) деятельности, продукты которой публикуются в виде отчетов, статей, монографий. В этом проявляется принципиальное отличие учебной и познавательной деятельности.

Деятельность осуществляется за счет выполнения действий, входящих в эту деятельность, составляющих ее. *Действие* — это единица деятельности. Так же, как и деятельность, действие направлено на какой-либо предмет, на достижение определенной цели и основано на каком-либо мотиве. Действие имеет и свою ориентировочную основу.

Так же, как и в деятельности в целом, в действии выделяют функциональные части, или стороны. П.Я. Гальперин [9] ввел в рассмотрение три таких части: *ориентировочную, исполнительную, контрольную.* Побуждающей силой любого действия является единый, или доминирующий, мотив всей деятельности, в состав которой входит данное действие. Так что определенная мотивация действия осуществляется в процессе реализации мотивационной части всей деятельности. Тем не менее, всегда надо стремиться к созданию «локальной» мотивации самого действия, и лучшим средством для этого является перевод действия в личный план обучаемого, создание условий, когда обучаемый считает выполнение действия своей личной обязанностью.

Важной является ориентировочная часть действия, которая во многом обеспечивает его успех. Собственно ориентировка обеспечивает выделение свойств и качеств объектов действия, которые существенны для их преобразования. Ориентировка направлена на выработку плана выполнения действия, на определение того, какие операции и в какой последовательности будут выполняться.

Исполнительная часть действия является одним из элементов исполнительной части деятельности в целом и представляет собой его рабочую часть, так как непосредственно обеспечивает преобразование объектов действия. Характер такого преобразования задается содержанием изучаемого предмета, и различают математические, физические и другие операции. При решении количественных задач (связанных с вычислениями) исполнительная часть, как правило, проста — это либо вывод формулы, либо вычисления по ней. Она намного сложнее и разнообразнее при выполнении различного вида лабораторных работ.

Контрольная часть направлена на проверку правильности результатов как ориентировочной, так и исполнительной части действия, на слежение за ходом выполнения действия, на проверку соответствия его намеченному

плану, на соотнесение продукта действия с его целью. В случае обнаружения ошибки, отклонения от правильного хода действия возникает необходимость исправления, коррекции.

Действия, выполняемые обучаемым в учебной деятельности, должны составлять систему действий, так как специфический результат их применения возникает не вследствие выполнения отдельных действий, а лишь в результате целостной их реализации в рассматриваемом наборе и последовательности.

Сократ в свое время заметил, что учитель не тот, кто дает, а тот, у кого берут. На первый взгляд, эта мысль кажется противоречивой, поскольку берут-то ученики; как они могут взять, если им не дадут? На самом деле никакого противоречия здесь нет. В процессе учения взять — значит совершить акт учебной деятельности. Умелый преподаватель организует учебную деятельность таким образом, чтобы обучаемый мог сам брать, т.е. правильно действовать: мыслить, предсказывать результаты деятельности, сопоставлять их с полученными, делать выводы. При этом обучающий должен не навязывать свое знание и свое понимание вопроса, а направлять и корректировать процесс мышления и действия обучаемых, не давая им отклоняться от цели, ориентируя их мыслительные поиски. Лучшим средством для этого является использование проблемного обучения.

Термин проблемное обучение в дидактике известен давно, он был употреблен еще в книге В. Бертона [3]. Второе рождение он получил в шестидесятые годы в трудах польского дидакта В. Оконя [45]. В СССР проблемное обучение развивалось и обосновывалось в работах российских исследователей А.В. Брушлинского, М.А. Данилова, В.И. Загвязинского, Т.В. Кудрявцева, И.Я. Лернера, А.М. Матюшкина, М.И. Махмутова, Н.А. Менчинской, Р.И. Малафеева, Л.В. Путляевой и др. С самого начала смысл термина проблемное обучение опирался на перевод с греческого слова проблема, что означает задача, вопрос. Понятие проблемной ситуации, так же как и понятие проблемы, стало основным для проблемного обучения, перекочевав из психологии в дидактику.

Общая схема проблемного обучения предполагает следующие четыре этапа: 1) постановка проблемы; 2) определение возможных путей ее решения; 3) выбор оптимального пути решения проблемы; 4) решение проблемы (М.И. Махмутов [42]).

Любая деятельность осуществляется путем решения задач, причем эти задачи должны быть специфическими для деятельности данного вида. В производственной, научно-исследовательской (научно-познавательной) деятельности результаты решения задач являются ее прямыми продуктами,

и, таким образом, сам факт решения задач соответствует целям деятельности. В учебной же деятельности решение задач — это не цель, но средство достижения целей, а именно, учебных целей. Другими словами, сам по себе результат решения учебных задач не представляет никакого интереса (единственное, что от него требуется, — это быть правильным). Важен процесс их решения, так как именно в процессе решения задач формируется способ действий.

Если же говорить о средствах, то в учебной деятельности особую роль играют так называемые средства регулирования деятельности. Сюда относятся знания об объектах деятельности и связях между ними, о способах преобразования объектов, о правилах выбора и последовательности применения требуемых преобразований, о способах контроля и оценки деятельности и т.д. Эти средства входят в содержание учебной деятельности. Вначале они становятся предметом усвоения, т.е. целью, а затем, будучи усвоены, превращаются в средства регулирования этой деятельности.

Содержательная часть является основой учебной деятельности, так как именно ради усвоения содержания и организуется эта деятельность. Содержание учебной деятельности должно проектироваться заранее. Это заданная система действий, подлежащих освоению, и знания, обеспечивающие освоение этой системы. Содержание образования должно соответствовать уровню современной науки, включать сведения, необходимые для создания у обучаемых представлений о частных и общенаучных методах познания и показывать им важнейшие закономерности процесса познания. В учебном процессе содержание образования является объектом как деятельности преподавателя, представляющего учебный материал (обучения), так и учебной деятельности (учения) обучаемого, усваивающего этот материал.

Выводы по главе 1

Web-системы дистанционного обучения в настоящее время активно исследуются и развиваются [20–39, 59–78, 82–92, 95–100, 103–112, 114–124, 126–131, 133–137, 140–150, 153–155]. Выгоды сетевого обучения ясны: аудиторная и платформенная независимости. Сетевое обучающее программное обеспечение один раз установленное и обслуживаемое в одном месте, может использоваться в любое время и по всему миру тысячами учащихся, имеющих любой компьютер, подключенный к Интернету. Дистанционное образование позволяет реализовать два основных принципа современного образования — «образование для всех» и «образование через всю жизнь».

Тысячи программ сетевого обучения и других образовательных приложений стали доступны в сети за последние годы и используются для предоставления дистанционных образовательных услуг, формирующих основу для создания единого открытого образовательного пространства. Проблема состоит в том, что большинство из них является не более чем статичными гипертекстовыми страницами и слабо поддерживает проблемный подход к обучению. Вместе с тем учебный процесс представляет собой совместную деятельность обучающего и обучаемого, который нельзя осуществить без решения задач, хотя в отличие от других видов деятельности (например, производственной или познавательной), где результаты решения задач являются ее прямыми продуктами, в учебной деятельности решение задач — это не цель, но средство достижения целей, а именно, учебных целей, направленных на изменение обучаемого, а не предметов внешнего мира.

Особенно важен проблемный подход при начальном обучении программированию, в процессе которого обучаемый должен овладеть навыками точного формулирования алгоритмов на языке высокого уровня. Что невозможно сделать, прочитав несколько руководств или прослушав курс лекций по программированию. Необходима практика конструирования алгоритмов, и здесь невозможно обойтись без подходящего набора примеров и задач, а также без оценки разработанных алгоритмов на правильность и качество.

ГЛАВА 2.
АДАПТИВНЫЕ МЕТОДЫ И СРЕДСТВА ДИСТАНЦИОННОГО ОБУЧЕНИЯ

В последнее время адаптивные гипермедиа-системы становятся все более и более популярными в дистанционном обучении, предоставляя управляемые пользователем средства доступа к информации. Активно ведутся исследования, направленные на создание продвинутых сетевых образовательных приложений, которые смогли бы обладать адаптивностью и интеллектуальностью [8, 15, 17, 21–22, 43, 54, 65, 66, 68].

Данная глава посвящена вопросам обеспечения персонализации при дистанционном обучении, особое внимание в ней уделяется анализу методов и средств адаптивной гипермедиа, используемых современными адаптивными обучающими Web-системами.

В разд. 2.1 определяются адаптивные гипермедиа-системы и дается ответ на вопрос, к чему они адаптируются, а также рассматриваются характеристики пользователя обучающей системы, важные для ее адаптации. Вопросу, что адаптируется в системах Web-обучения, посвящен разд. 2.2. В разд. 2.3 рассматриваются такие технологии, как выстраивание программы обучения, интеллектуальный анализ решений обучаемого, поддержка в решении задач, привнесенные в обучающие Web-системы из несетевых интеллектуальных обучающих систем. Разд. 2.4 и 2.5 содержат описание методов и технологий адаптации содержания и навигации, характерных для адаптивных гипермедиа-систем. Моделям предметной области, пользователя и адаптации, используемым в адаптивных гипермедиа-системах, посвящен разд. 2.6. Разд. 2.7 содержит примеры существующих экспериментальных адаптивных обучающих Web-систем. Завершают изложение выводы по главе 2.

2.1. Адаптивные гипермедиа-системы

Поддержка адаптивных методов в гипермедиа-системах оказывается весьма полезной в тех случаях, когда имеется *одна* система, обслуживающая *множество* пользователей с *различными* целями, уровнем знаний и опытом, *и* когда лежащее в ее основе гиперпространство является относительно *большим* [68]. Поэтому области применения адаптивной-гипермедиа выходят далеко за границы обучающих систем [7, 8, 21, 65, 113, 122, 126, 139]. Вместе с тем обучающие гипермедиа-системы, в которых пользователь или ученик имеет конкретную цель обучения (включая и такую цель, как общее образование), являются типичным приложением адаптивных

гипермедиа-систем. В этих системах основное внимание уделяется знаниям обучающихся, которые могут сильно различаться. Состояние знаний изменяется во время работы с системой. Таким образом, корректное моделирование изменяющегося уровня знаний, надлежащее обновление модели и способность делать правильные заключения на базе обновленной оценки знаний являются важнейшей составляющей обучающей гипермедиа-системы.

По возможностям адаптации различаются следующие типы гипермедиа-систем [8, 21, 22, 83]:

1. *Адаптированные (приспособленные) гипермедиа-системы* (adapted hypermedia systems) — в них адаптация привносится в систему самим разработчиком после фазы тестирования. В этом случае адаптация не может быть корректной для каждого отдельного пользователя.

2. *Адаптируемые (приспосабливаемые) гипермедиа-системы* (adaptable hypermedia systems) могут модифицироваться только по явному требованию пользователя. Адаптируемые системы позволяют пользователю явно устанавливать предпочтения или предоставляют профиль через заполнение формы. Вся информация, предоставленная пользователем, хранится в его модели, которая модифицируется только по явному запросу пользователя. Представление информации затем адаптируется к этой модели. Некоторые системы могут иметь очень сложные модели пользователя, в то время как другие различают только несколько стереотипных пользователей типа «начинающего», «среднего» и «эксперта».

3. *Адаптивные гипермедиа-системы* (adaptive hypermedia systems) сами могут адаптироваться к потребностям пользователя. Адаптивные системы формируют модель пользователя, отслеживая его навигацию в информационном пространстве, а также посредством специальных тестов (в системах обучения). Представление адаптируется к модели пользователя, которая постоянно обновляется, по мере того как пользователь просматривает информацию.

Класс адаптивных гипермедиа-систем состоит из всех таких гипертекстовых и гипермедиа-систем, которые отражают некоторые особенности пользователя в модели пользователя и применяют эту модель для адаптации различных видимых для пользователя аспектов системы. Адаптивная гипермедиа-система собирает информацию о пользователях и на основе их индивидуальных характеристик она адаптирует свое содержание и навигационные возможности к потребностям конкретного пользователя (рис. 2.1).

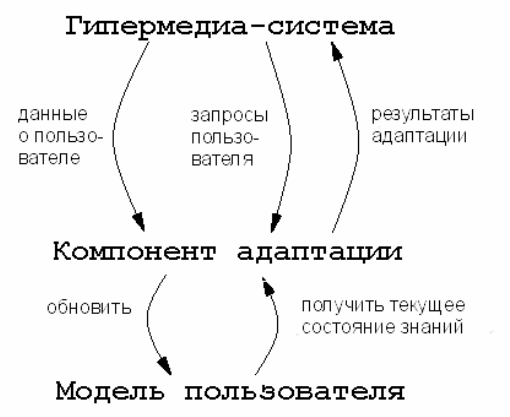


Рис. 2.1. Схематическое представление адаптивной гипермедиа-системы

Первый вопрос, возникающий при рассмотрении модели конкретной адаптивной системы, состоит в следующем: какие характеристики пользователя могут быть приняты во внимание для обеспечения адаптации? К каким параметрам (различным для различных пользователей и, возможно, различным для одного и того же пользователя в разные моменты времени) система может адаптировать свое поведение? Вообще, существует довольно много параметров, связанных как с конкретным видом текущего использования системы, так и с пользователем как таковым, которые адаптивная система может принять во внимание [18, 21, 22, 65, 66, 168]: знания, цели, предпочтения, подготовку, опыт и скорость обучения пользователя.

Цель пользователя (или *задача пользователя*) — это параметр, зависящий, в большей степени, от самой природы текущего использования гипермедиа-системы, нежели от пользователя, работающего с системой, как такового. Цель пользователя — это наиболее изменчивая его характеристика. Она почти всегда будет другой при новом сеансе работы с системой, а иногда может изменяться и во время одного сеанса работы. В обучающих системах целесообразно различать *локальные*, или цели *нижнего уровня*, которые могут изменяться достаточно часто, и *общие*, или цели (задачи) *высокого уровня*, являющиеся более стабильными. Например, цель изучения курса или некоторой темы курса — это цель высокого уровня, сохраняющаяся в течение нескольких сеансов, в то время как цель решения зада-

чи — цель нижнего уровня, она может изменяться от одной учебной задачи к другой несколько раз за один сеанс работы.

В существующих в настоящее время адаптивных гипермедиа-системах уровень **знаний пользователя** является наиболее важной информацией для адаптации. Система должна постоянно обновлять свою оценку уровня знаний пользователя, и адаптационная компонента должна использовать реальное состояние знаний для выполнения шагов адаптации.

Уровень подготовки пользователя и **имеющийся у него опыт работы** с данной гипермедиа-системой — это две характеристики, имеющие нечто общее с уровнем знаний пользователя, но функционально отличающиеся от него.

Важной характеристикой, рассматриваемой системами адаптивной гипермедиа, является **набор (система) предпочтений пользователя**. По различным причинам пользователь может предпочитать одни узлы, ссылки и части страниц гипермедиа другим. Эти предпочтения могут быть абсолютными или относительными, то есть зависящими от текущего узла, цели или текущего контекста вообще.

Другая важная группа характеристик связана с определением пользователя как индивидуума. Примерами **личностных характеристик** являются *персональные факторы* (например, интроверт или экстраверт), *когнитивные факторы* (например, *скорость обучения*) и *стили обучения*.

Поскольку пользователи одного и того же серверного Web-приложения могут находиться виртуально повсеместно и могут использовать различное оборудование, важной задачей становится адаптация к **пользовательской среде**.

Некоторые авторы (см., например, [13]) помимо адаптации к отдельному пользователю рассматривают еще адаптации к множествам пользователей, выделяя при компьютерном обучении три иерархических уровня адаптации к обучаемым: (1) адаптация к студентам как категории пользователей; (2) адаптация к группе студентов; (3) адаптация к отдельному студенту.

Первый уровень адаптации предусматривает адаптацию к каждой категории пользователей компьютерной системы обучения в зависимости от их потребностей и обычно реализуется созданием специального интерфейса для каждого выделенного класса. Такой подход характерен для любых компьютерных систем. В интеллектуальных обучающих системах учащемуся необходимо предоставить следующие возможности: обучение, проверка знаний, упражнения, помощь и справочная информация, видеолекции и их презентации, вопросы преподавателю, конференции, студенческие форумы,

электронные методические пособия, ввод комментариев по ходу занятия и др.

Адаптация к группе студентов обеспечивает настройку системы в зависимости от выбранной специальности, образовательной программы, возраста и психологической направленности личности. Этот уровень адаптации базируется, в первую очередь, на решении двух основных вопросов дидактики: «чему учить?» и «как учить?». Ответ на первый вопрос определяет цели обучения, т.е. объем необходимых знаний, умений и навыков и степень их освоения. Решение второго вопроса дидактики («как учить?») обуславливает выбор методов обучения, наиболее подходящих для группы учащихся, а также способов представления информации. На выбор методов обучения и способов представления информации влияют как возраст обучаемого, так и психологическая направленность его личности (ориентация на себя, на задачу, на взаимодействие).

2.2. Что адаптируется в системах Web-обучения?

Сетевые обучающие системы успешно объединяют технологии адаптации, используемые в интеллектуальных обучающих системах и адаптивных гипермедиа-системах [8, 11, 21, 22, 48, 49, 57, 65, 66, 68].

Целью различных интеллектуальных обучающих систем является использование знаний о сфере обучения, обучаемом и о стратегиях обучения для обеспечения гибкого индивидуализированного изучения и обучения. Для достижения этого ими традиционно используются следующие основные технологии [63, 70, 73, 123, 130]: построение последовательности курса обучения, интеллектуальный анализ ответов обучаемого и интерактивная поддержка в решении задач. К группе технологий интеллектуальных адаптаций сетевых обучающих систем следует отнести также технологию, получившую название *подбора моделей обучаемых* (или просто *подбор моделей* [63]). Суть ее состоит в анализе и подборе модели для многих обучаемых одновременно, и поэтому она почти не имеет корней в несетевых (доинтернетовских) образовательных системах, поскольку обычные адаптивные и интеллектуальные образовательные системы работают с одним обучаемым (и одной моделью обучаемого) за раз.

Что касается гипермедиа-систем, то в них область адаптации весьма ограничена и существует не так уж много параметров, которые можно изменять. С общей точки зрения гипермедиа-система состоит из набора узлов или *гипердокументов* (для краткости, будем называть их "страницами"), связанных ссылками. Каждая страница содержит некоторую локальную

информацию и несколько ссылок на релевантные страницы. Системы гипермедиа могут также содержать индексную структуру и глобальную карту, которые обеспечивают доступ по ссылкам ко всем возможным страницам. Поэтому адаптация в гипермедиа-системе может состоять в настройке содержания очередной страницы (*адаптация на уровне содержания*) или в изменении ссылок с очередной страницы, индексных страниц и страниц карт (*адаптация на уровне ссылок*). Поэтому различаются адаптации на уровне содержания и на уровне ссылок как два различных класса гипермедиа-адаптации, первый из которых называется *адаптивным представлением* (adaptive presentation), а второй — *адаптивной поддержкой навигации* (adaptive navigation support).

2.3. Технологии интеллектуальных обучающих систем

Цель технологии *выстраивания последовательности обучения* (curriculum sequencing), также называемой *технологией учебного планирования* (instructional planning technology), — помочь обучаемому найти свой «оптимальный путь» через учебный материал.

Различают два вида выстраивания: активный и пассивный. *Активное* выстраивание подразумевает наличие *цели обучения* (подмножество понятий или тем, которыми надо овладеть). Системы с активной последовательностью могут построить лучший индивидуальный путь для достижения цели. *Пассивное* выстраивание (которое также называется *корректировкой*) начинает действовать тогда, когда пользователь не способен решить задачу или ответить правильно на вопрос (вопросы). Цель системы с пассивным выстраиванием — предложить студенту подмножество доступного материала для изучения, которое может заполнить пробел в его знаниях для разрешения непонимания. В системах с активной последовательностью различают системы с жесткой и приспосабливаемой целью обучения.

Также разделяют два уровня выстраивания: высокий и низкий. Выстраивание *высокого уровня* (или *выстраивание знаний*) определяет следующую подцель для изучения: понятие, набор понятий, тему или урок, которые необходимо выучить на следующем шаге. Выстраивание *низкого уровня* (или *выстраивание заданий*) определяет следующее учебное задание (задачу, пример, тест) внутри текущей подцели.

Технология *интеллектуального анализа решений обучаемого* имеет дело с окончательными ответами студента на обучающие задания (которые могут колебаться от простых вопросов до сложных задач программирования) без разъяснения причин, по которым ответ был получен. В отличие от

не-интеллектуальных проверяющих программ, которые не могут сказать ничего кроме того, правильный ответ или нет, интеллектуальные анализаторы могут подсказать обучаемому, что именно неправильно или неполно и какие отсутствующие или неверные знания ответственны за ошибку.

В течение многих лет осуществление *поддержки в решении задач* рассматривалось как главная обязанность интеллектуальных обучающих систем и их основное достоинство. Интеллектуальные обучающие системы используют для этого интеллектуальный анализ решений обучаемого, интерактивную поддержку решения задач и поддержку решения задач на примерах.

Технология *интерактивной поддержки в решении задач* вместо ожидания конечного решения предоставляет обучаемому интеллектуальную помощь на каждом шаге решения задачи. Уровень помощи может быть разным: от оповещения о неправильно сделанном шаге до выдачи совета и выполнения следующего шага за студента. Системы (часто называемые *интерактивными тренажерами*), в которых реализуется эта технология, могут наблюдать за действиями студента, понимать их и использовать это понимание для предоставления помощи и обновления модели обучаемого.

Технология *поддержки в решении задач на примерах* помогает обучаемым решать новые задачи, не выделяя их ошибки, а предлагая примеры успешного решения схожих задач из их более раннего опыта (это могут быть примеры, объясненные им, или задачи, решенные ими ранее).

Используются два вида *подбора моделей обучаемых*: адаптивная поддержка сотрудничества и интеллектуальное наблюдение за классом. Эти виды полностью отличаются друг от друга и рассматриваются как различные технологии внутри группы подбора моделей обучаемых [63]. Целью *адаптивной поддержки сотрудничества* является использование знаний системы о разных обучаемых для подбора групп в разных видах сотрудничества (см., например, [76, 81]). *Интеллектуальное наблюдение за классом* также основано на возможности сравнивать записи о разных обучаемых, но вместо поиска совпадений ищутся различия. Цель сравнения — определение тех обучаемых, которые учатся существенно отличающимся образом от своих сокурсников, например, развиваются слишком быстро или слишком медленно или просто осуществили доступ к гораздо меньшему материалу, чем остальные. Эти обучаемые нуждаются в большем внимании преподавателя, чем остальные. Причем это внимание различно, поскольку преподаватель должен: (а) бросить вызов тем, кто может; (б) дать больше объяснений тем, кто не может; (в) подтолкнуть тех, кто мешкает.

2.4. Адаптация содержания в гипермедиа-системах

Основная идея всех технологий адаптации на уровне содержания состоит в том, чтобы адаптировать представление страницы, к которой обращается пользователь, к его текущему уровню знаний, целям и другим характеристикам.

Адаптивное представление текста позволяет изменить текстовое содержание гипермедиа-страницы в зависимости от модели пользователя. В гипермедиа-системах страница может содержать не только текст, как в классических гипертекстовых системах, но и мультимедиа-информацию, что требует *адаптации мультимедиа. Адаптация модальности* — новая технология адаптации содержания высокого уровня; она позволяет осуществить выбор тех дополнительных средств (музыка, видео, речь, анимация и т.д.), которые наиболее релевантны пользователю для данного узла.

Можно выделить следующие цели (методы) адаптации на уровне содержания.

1. *Дополнительные объяснения* (additional explanations). Только те части документа должны выдаваться пользователю, которые соответствуют уровню его знаний или цели.

2. *Предварительные объяснения* (prerequisite explanations). Цель — снабдить пользователя необходимой ему информацией для понимания документа. По его модели проверяются предварительные знания, требуемые для понимания содержания страницы. Если пользователь не обладает какими-то знаниями, соответствующая информация должна быть добавлена к странице.

3. *Сравнительные объяснения* (comparative explanations). Идея сравнительных объяснений состоит в пояснении новых тем путем подчеркивания их связей с уже известными темами.

4. *Варианты объяснений* (explanation variants). Предоставляя различные объяснения для некоторых частей документа, мы можем выбрать те из них, которые максимально подходят для данного пользователя. Этот подход является расширением метода предварительных объяснений.

5. *Сортировка* (sorting). Различные части документа сортируются в соответствии с их актуальностью для пользователя.

Для достижения целей адаптации на уровне содержания используются следующие техники.

1. *Условный текст* (conditional text) предполагает, что вся информация о понятии разбивается на определенные текстовые куски. Для каждого из

этих кусков определяется требуемый уровень знаний пользователя, при наличии которого производится его вывод.

2. *Эластичный текст* (stretchtext). Технология более высокого уровня, которая может также включать и выключать различные части текста в соответствии с уровнем знания пользователя, когда некоторые ключевые слова документа могут заменять более длинные описания или заменяться ими, если этого требует реальный уровень знаний обучаемого.

3. *Варианты страниц* (page variants) *и варианты фрагмента* (fragment variants) — техники, позволяющие реализовывать метод вариантного объяснения. Например, система хранит несколько вариантов объяснения каждого понятия (варианты фрагментов), и пользователь получает страницу, содержащую варианты, соответствующие его знанию о представленных понятиях. Технологии вариантов страницы и вариантов фрагмента могут быть скомбинированы, например, для обеспечения адаптации и к подготовке, и к знанию пользователя. Текущий вариант страницы для конкретного узла выбирается согласно подготовке пользователя. Эта страница затем может быть адаптирована: для каждого понятия, представленного на странице, система выбирает объяснение, которое наиболее соответствует уровню знаний пользователя.

4. *Технология, основанная на фреймах* (frame-based technique), является наиболее мощной из всех технологий адаптации содержания. При ее использовании вся информация об отдельном понятии представлена в форме фрейма. Слоты фрейма могут содержать несколько вариантов объяснения понятия, связи с другими фреймами, примерами и т.д.

2.5. Адаптация навигации в гипермедиа-системах

Используя адаптацию на уровне ссылок, можно персонализовать возможности навигации по гипермедиа-системе с учетом целей, знаний и других характеристик индивидуального пользователя. Адаптация на уровне ссылок ограничивает количество ссылок и, следовательно, объем навигационных возможностей. Она полезна для исключения ситуации «затерянности в гиперпространстве». Можно выделить следующие пять целей адаптации навигации.

1. *Глобальное руководство* (global guidance), помогающее пользователю найти самый короткий путь к информационной цели с минимальным блужданием.

2. *Локальное руководство* (local guidance), которое помогает пользователю сделать один навигационный шаг, предлагая ссылки от текущего узла, наиболее подходящие для перехода.

3. *Поддержка локальной ориентации* (local orientation support), позволяющая помочь пользователю понять, что находится вокруг и какова его относительная позиция в гиперпространстве.

4. *Поддержка глобальной ориентации* (global orientation support), которая помогает пользователю понять структуру всего гиперпространства и свою абсолютную позицию в нем.

5. *Управление индивидуализированными представлениями* (managing personalized views), позволяющее организовать электронное рабочее место пользователям, нуждающимся в доступе к достаточно небольшой части гиперпространства для своей повседневной работы.

Технологии адаптивной навигационной поддержки могут быть классифицированы согласно способу, который они используют для адаптации представления ссылок. Используются следующие шесть технологий.

1. *Полное руководство* (direct guidance) — наиболее простая технология адаптивной навигационной поддержки. Различаются два метода полного руководства: метод *«следующего наилучшего»* (next best), предполагающий предоставление кнопки «next» для дальнейшей навигации через гиперпространство, и метод *установления последовательности страниц,* или *следа* (page sequencing or trails), осуществляющий генерирование последовательности страниц для просмотра всей гипермедиа-системы или ее части.

2**.** *Адаптивная сортировка (упорядочение) ссылок* (adaptive sorting (ordering) of links) — технология, которая вместо одной «наилучшей» ссылки предоставляет список ссылок, упорядоченный по убыванию релевантности, в соответствии с моделью пользователя. Различаются два метода сортировки: метод *сортировки по подобию* (similarity sorting), когда ссылки сортируются, исходя из их подобия данной странице, и метод *предварительных знаний* (prerequisite knowledge), когда ссылки упорядочиваются согласно предварительно необходимым знаниям.

3. *Адаптивное сокрытие ссылок* (adaptive link hiding) — в настоящее время наиболее часто используемая технология для адаптивной навигационной поддержки. Идея навигационной поддержки с помощью сокрытия состоит в том, чтобы ограничить навигационное пространство, скрывая ссылки к «нерелевантным» страницам. Различают следующие три вида сокрытия ссылок: *сокрытие ссылок* (link hiding), *удаление ссылок* (link removal) и *отключение ссылок* (link disabling).

4. *Адаптивное аннотирование ссылок* (adaptive link annotation) — технология, смысл которой состоит в пополнении ссылок некоторыми комментариями, чтобы дать пользователю подсказку относительно содержания страницы за аннотируемыми ссылками. Аннотации могут быть представлены в текстовой форме или в форме визуальных подсказок, например, используя различные значки, цвета или размеры шрифтов. Наиболее популярный способ аннотирования ссылок — использование *метафоры светофора* (traffic light metaphor).

5. *Адаптивное генерирование ссылок* (adaptive link generation). Генерирование ссылок включает три случая: обнаружение новых полезных ссылок между документами и добавление их к постоянному набору существующих ссылок, генерирование ссылок для навигации между элементами, основанной на подобии, и динамическая рекомендация релевантных ссылок.

6. *Адаптация карты* (map adaptation) — технология, включающая различные пути адаптации формы глобальных и локальных гипермедиа карт (графические представления структуры ссылок), предоставляемых пользователю.

2.6. Архитектура адаптивной гипермедиа-системы

На абстрактном уровне в любой адаптивной гипермедиа-системе функционально можно выделить следующие три основные взаимодействующие компоненты [8, 21, 22, 91, 155]: *модель предметной области* (domain model), которая описывает, как структурировано информационное содержимое приложения (или гипердокумента); *модель пользователя* (user model), представляющая пользовательские предпочтения, знания, цели, историю навигации и другие относящиеся к пользователю характеристики; *модель обучения* (teaching model), называемая также *моделью адаптации* (adaptation model), позволяющая осуществить необходимые адаптации системы, например, с помощью так называемых *педагогических правил* (pedagogical rules) или *правил адаптации* (adaptation rules).

Помимо указанных трех моделей адаптивная гипермедиа система может содержать так называемый *механизм адаптации* (adaptive engine). Это — реальный программный продукт, являющийся частью системы, который используется ею для конструирования и адаптации содержимого и ссылок. Механизм поддерживает некоторую библиотеку функций для конструирования информационных страниц из фрагментов, основанных на элементах из моделей предметной области, пользователя и адаптации. Язык (обычно довольно простой) применяется с целью выбора «конструктора» для ис-

пользования. Некоторые механизмы адаптации могут поддерживать способ определения новых конструкторов или расширения существующих. Однако, достаточно мощный механизм должен поддерживать достаточно стандартную функциональность для облегчения потребностей авторов явно специфицировать новые конструкторы в большинстве приложений. Механизм адаптации также изменяет пользовательскую модель, отслеживая поведение пользователя и, таким образом, принимая во внимание, как изменяются его знания.

Модель предметной области дает описание предметной области на концептуальном уровне и представляет собой совокупность объектов (концептов и межконцептных отношений), каждый из которых имеет уникальное имя.

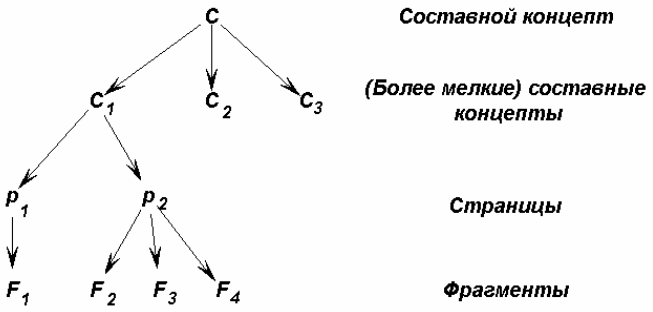


Рис. 2.2. Пример иерархии концептов

Концепты — это абстрактные объекты, используемые для представления элементов информации предметной области. Существуют концепты *нижнего уровня* (*атомарные концепты* или *фрагменты*), каждый из которых соответствует одному фрагменту информации, и концепты *высшего уровня* (или *составные концепты*), состоящие из множества других концептов [89, 91, 153–155]. Атомарные концепты являются первичными в модели и могут не подвергаться адаптации. Их атрибутные и анкерные значения принадлежат «внутрикомпонентному уровню» и таким образом являются зависимыми от реализации и не описываемыми в модели. Составные концепты имеют два специальных атрибута: последовательность детей (концептов) и функция конструктора (для обозначения, как дети соединя-

ются). Если все дети некоторого концепта атомарны, то такой концепт называют *страницей* (Рис. 2.2).

Межконцептные отношения представляют различные отношения между двумя или более концептами. Например, часто используются следующие типы бинарных отношений между парами концептов: тип *часть* (part-of) — это композиционное отношение, указывающее на то, что первый концепт является частью второго; тип *связь* (link) представляет существование связи первого концепта со вторым (например, в виде гиперссылки); тип *предпосылка* (prerequisite) означает, что первый концепт должен быть предварительно изучен до изучения второго концепта; тип *блокиратор* (inhibitor) говорит о том, что первый концепт не должен изучаться до тех пор, пока не будет изучен второй (Рис. 2.3).

Межконцептные отношения могут иметь атрибуты. Например, приписывая некоторое вещественное число из интервала [0,1] тому или другому бинарному отношению можно указать: для отношения типа *часть* — какую часть второго концепта представляет первый концепт; для отношения типа *предпосылки* — как много знаний о первом концепте должно быть у пользователя, чтобы информация о втором концепте становилась желаемой; для отношения типа *блокировки* — какую границу знаний о первом концепте должны превысить знания пользователя, чтобы стало нежелательным получение знания о втором концепте.

Можно выделить три типа моделей предметной области, различающихся по уровню сложности структуры [66]. Самую простую структуру имеет модель *первого уровня*, являющаяся независимым множеством концептов (отсутствуют межконцептные отношения). Модель *второго уровня* (сетевая модель предметной области) предполагает наличие связей между концептами и представляет собой семантическую сеть, состоящую из концептов и межконцептных отношений. Может существовать несколько типов концептов и межконцептных отношений. Модель *третьего уровня* (фреймовая модель предметной области) предполагает наличие у концептов внутренней структуры в виде множества атрибутов, при этом концепты различных типов могут иметь различные множества атрибутов.

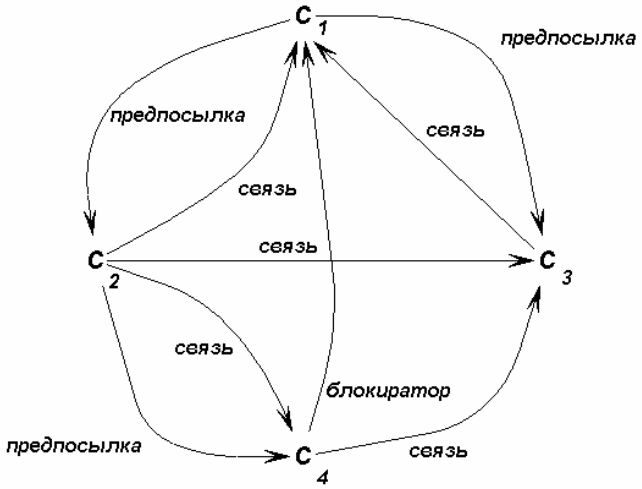


Рис. 2.3. Пример структуры межконцептных отношений

Одной из самых важных функций модели предметной области является обеспечение структуры для представления знаний пользователя системой.

Адаптивные гипермедиа-системы можно подразделить на три основные группы в зависимости от метода организации связи между моделью предметной области (концептами) и гиперпространством системы (гипермедиа-страницами). Самый простой метод — это *индексация страниц* концептами, относящимися к содержимому этих страниц. Второй метод, похожий на предыдущий, это — *индексация фрагментов*: содержимое страницы разбивается на множество фрагментов, каждый из которых отдельно индексируется множеством концептов, относящихся к содержимому данного фрагмента. Третий метод (*прямая связь*) отличается от предыдущих методов тем, что для страниц не поддерживаются индексы; гиперпространство адаптивной гипермедиа-системы строится непосредственно исходя из структуры модели предметной области. Таким образом, каждый концепт модели представлен гипермедиа-страницей или гипердокументом, а отношения между концептами соответствуют гиперссылкам между страницами. Страница или документ, представляющий концепт, могут быть как статическими, так и динамическими: т.е. генерироваться на лету, исходя из внутренней структуры концепта.

Модель пользователя адаптивной гипермедиа-системы предполагает *явное* представление знаний, предпочтений, целей, интересов, истории навигации и других характеристик пользователя и служит для адаптации к нему различных аспектов адаптивной гипермедиа-системы [8, 21, 22, 96, 117]. Модель пользователя состоит из именованных элементов, для которых хранится набор пар вида атрибут-значение (компонентов модели пользователя). На концептуальном уровне можно представлять ее в виде табличной структуры, в которой для каждого элемента хранятся значения атрибутов. Большинство элементов в модели пользователя представляют концепты модели предметной области. Некоторые другие элементы могут кодировать различные аспекты пользователя, такие как цели, предпочтения, интересы или стереотипную классификацию (типа новичок, эксперт) и т.д. [89, 91, 153–155].

Можно классифицировать модели пользователей согласно следующим основным свойствам: способ получения информации (явный или неявный), степень специализации модели (общие или индивидуальные модели), модифицируемость модели (статические или динамические модели), временная протяженность (краткосрочные или долгосрочные модели), метод использования модели (дескриптивные или прескриптивные модели).

Различаются два основных подхода к моделированию пользователя: моделирование перекрытий и моделирование стереотипного пользователя.

Оверлейное моделирование или *моделирование перекрытий* (*overlay modeling*) [98] чаще всего используется в интеллектуальных системах обучения для моделирования знаний, при этом знания пользователя описываются как подмножество знаний эксперта в данной области, отсюда сам термин «перекрытие» («оверлей»). Недостаток знаний обучающегося выводится посредством сравнения их со знаниями эксперта. Для каждого концепта модели предметной области в модели знаний пользователя вычисляется и сохраняется некоторое значение (или несколько значений), оценивающее уровень знания пользователем этого концепта (оверлейная модель). *Оверлейная модель* знаний (*overlay model*) может быть представлена как множество пар «концепт — значения атрибутов» [153–155].

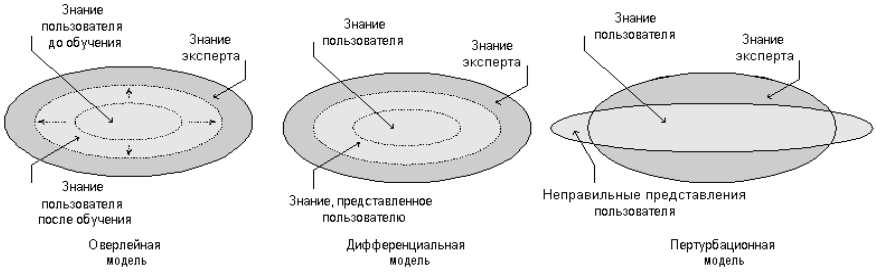


Рис. 2.4. Виды моделей знаний при оверлейном моделировании

Таким образом, в рамках оверлейной модели предполагается, что знание пользователя составляет некоторое подмножество знания эксперта, и цель обучения состоит в расширении этого подмножества. Модель также предполагает, что пользователь не будет изучать того, чего не знает эксперт. В частности, не принимаются во внимание неправильные представления и заблуждения, изначально имеющиеся у пользователя, или приобретенные им в процессе обучения. Второй недостаток оверлейной модели заключается в том, что нет механизма для разграничения знаний, которые пользователь еще не приобрел, и знаний, которые еще не были ему представлены, что имеет смысл для стратегии обучения.

Дифференциальная модель (Differential model) является расширением оверлейной модели. В ней знания эксперта разделены на те, которые уже были представлены пользователю, и те, которыми пользователь еще не обязан обладать, а простая оверлейная модель применяется лишь к знаниям, уже представленным пользователю. Аналогично оверлейной, дифференциальная модель не принимает во внимание неправильные представления и ошибки пользователя.

Пертурбационная модель (Perturbation model) принимает во внимание те знания, которыми может обладать пользователь вне знаний эксперта. Пертурбационная модель расширяет модель эксперта добавлением библиотеки ошибок (bug library). Процесс ее создания может быть перечисляющим или порождающим. Перечисляющий процесс составляет список всех возможных неправильных представлений с помощью анализа предметной области и ошибок, которые допускает пользователь. Порождающий метод пытается генерировать ошибки исходя из лежащей в основе познавательной теории. Оверлейная модель может быть применена поверх комбинированной модели эксперта и библиотеки ошибок. Как и для простой оверлей-

ной модели, цель обучения — увеличить подмножество знания эксперта при исключении неправильных представлений.

Стереотипное моделирование (*stereotype modeling*) — один из первых методов в области моделирования, классифицирующий пользователей по стереотипам [135]. Предполагается, что пользователи, относящиеся к одному классу, имеют одни и те же характеристики. При использовании стереотипной модели иногда полезно различать два типа стереотипного моделирования: фиксированное моделирование и моделирование по умолчанию.

Стереотип в общем случае состоит из следующих частей: множества инициирующих условий (триггеров), являющихся логическими выражениями, активирующими стереотип; множества условий отвода (ретракций), ответственных за деактивацию активного стереотипа; множества предположений (выводов) стереотипа, служащих предположениями по умолчанию при связывании пользователя со стереотипом.

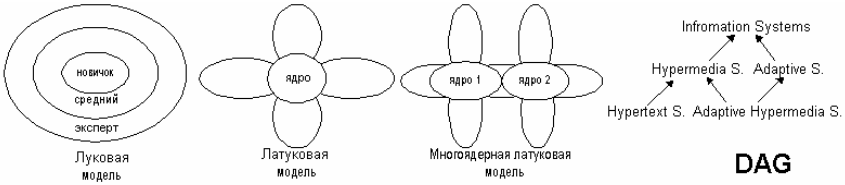


Рис. 2.5. Виды моделей знаний при стереотипном моделировании

Выделяют четыре модели согласно типу организации связи между стереотипами (Рис. 2.5). *Многоуровневая* (или *луковая*) *модель* (*onion model*) — это иерархическая модель, где содержимое стереотипов линейно упорядочено по отношению быть подмножеством. *Летисная* (*латуковая* или *салатная*) *модель* (*lettuce model*) характеризуется наличием стереотипа-ядра, содержимое которого является подмножеством содержимого всех других стереотипов, в остальном независимых друг от друга. *Многоядерная летисная* (*многоядерная латуковая* или *многоядерная салатная*) *модель* (*multikernel lettuce model*) является обобщением летисной модели, допускающей существование нескольких ядер, являющихся пересечениями некоторых стереотипов. *Ориентированный ациклический граф* (*DAG*[1]) является

[1] Здесь и далее мы используем без определения стандартные термины теории графов (см., например, [12] или [19]).

обобщением многоядерной модели, допускающей существование общих частей у двух или более ядер, также представляющихся ядрами, и т.д.

Модель адаптации описывает, как должна происходить адаптация в зависимости от моделей пользователя и предметной области. Она состоит из правил адаптации, которые формируют связь между моделью предметной области и моделью пользователя и определяют представление генерируемой информации и обновление пользовательской модели [89, 91, 153–155].

Обычно правило содержит две основные части: *условие*, при котором происходит срабатывание данного правила, и *трансформация* — описание того действия, которое задается данным правилом. Условие может предполагать возникновение некоторого внешнего события (такие правила называются ECA-правилами, в отличие от CA-правил, в которых условие — это просто логическое выражение), например, обращения к странице, а также истинность некоторого логического выражения, построенного над значениями атрибутов из моделей пользователя и предметной области и проверяемого в те моменты, когда событие возникает. Действие может состоять в модификации значений атрибутов в модели пользователя или в присваивании объекту спецификации представления. Также в правиле может быть «фаза» выполнения, указывающая на момент времени, в который должно применяться правило: до или в течение генерации (фаза “pre”) и после генерации страницы (фаза “post”). Также в правиле может быть определено, может оно инициировать запуск других правил или нет.

В качестве примера можно привести два простых правила, написанных с использованием произвольно выбранного синтаксиса.

Например, следующее правило определяет, что при обращении к странице для соответствующего концепта во время фазы “post” в модели пользователя устанавливается атрибут «чтение», равный истине:

<access(C) => C.read:= true; post; true>

Правило также утверждает, что оно запустит другие правила, которые имеют атрибут «чтения» (C.read:= true) в своей левой части.

Другой пример: следующее правило выражает, что когда пользователь обращается к странице, определяющей концепт, «готовый для чтения», то значение знания для этого концепта становится «изученным» в фазе “pre”:

<(access(C) & C.ready-to-read = true) =>
C.knowledge-value:=learned; pre; true>

Было показано [60], что многие из реальных приложений используют такие ECA-правила, в которых логические выражения принимают истинные значения в точности тогда, когда возникают события, составляющие

условия данных правил; такие правила получили название квази-CA-правил.

Все правила адаптации подразделяются [154] на *родовые* или *обобщенные* правила (generic rules) и *специфические* или *конкретные* правила (specific rules). В отличие от обобщенных правил, которые применимы ко всем концептам и всем межконцептным отношениям некоторого заданного типа и используют связанные переменные для представления в них концептов и межконцептных отношений, специфические правила описывают преобразования для конкретных концептов, множеств концептов и межконцептных отношений и не используют переменных. Специфические правила имеют приоритет над обобщенными правилами, и, таким образом, они используются для определения исключений в общих правилах.

Одной из важных проблем, возникающих при построении модели адаптации, является обеспечение таких свойств систем правил адаптации, как нетеровость и завершаемость [19], позволяющих не следить за порядком применения правил адаптации (с помощью механизма адаптации).

2.7. Примеры адаптивных обучающих гипермедиа-систем

Много интересных адаптивных гипермедиа-систем для обучения было разработано в начале 90-х годов прошлого столетия. Интерес, вызванный развитием дистанционного Web-обучения, заметно усилил эти исследования, увеличив количество и типы разрабатываемых систем. Все первые системы были в значительной степени лабораторными, построенными исключительно для исследования тех или иных методов адаптации в контексте обучения. В отличие от них, ряд более современных систем обеспечивает среды и авторизованные инструменты для разработки Web-курсов. Появление большого числа авторизованных инструментов демонстрирует зрелость адаптивной обучающей гипермедиа и является ответом на Web-спровоцированный запрос на создание адаптивных к пользователям дистанционных учебных курсов.

Выбор Web в качестве платформы разработки адаптивных гипермедиа стал стандартом. Этот выбор дал долгую жизнь ряду адаптивных учебных гипермедиа-систем, разработанных до 1996 г., таких как ELM-ART, InterBook и 2L670. Эти системы были существенно модифицированы после 1996 г., расширены большим числом новых технологий и использованы для ряда экспериментальных исследований. Не удивительно, что почти все адаптивные учебные гипермедиа-системы, разработанные в указанный период, являются Web-системами.

Система ELM-ART [73] и ее «наследники» ELM-ART II [150] и INTERBOOK [72] были одними из первых адаптивных гипермедиа-систем, используемых в Интернете. Они основаны на отдельной системе ELM-PE [148], вводном курсе по программированию на LISP. Ее авторы используют *эпизодическую модель ученика* (ELM) [149] для диагностики полных и неполных решений проблем. Эпизодическая модель хранит информацию о пользователе в виде коллекции *эпизодов*. Эти эпизоды могут сравниваться с прецедентами в основанном на прецедентах обучении [140]. Для построения модели ученика код написанной им программы анализируется в терминах области знаний, с одной стороны, и описания задачи, с другой. Эта диагностика дает в итоге дерево вывода понятий и правил, которые ученик мог использовать при написании программы.

В ELM-ART понятия связаны друг с другом посредством предварительных условий и результатов. Таким образом строится понятийная сеть. Наблюдения за пользователем производятся посредством мониторинга посещенных страниц: понятие, соответствующее посещенной странице, помечается в понятийной сети как известное пользователю. Для аннотации ссылок авторы используют метафору светофора. Красный кружок обозначает страницы, для изучения которых пользователю недостает знаний; зеленый обозначает страницы, на которые предлагается обратить внимание, и т.д. ELM-ART также содержит интерактивные примеры, которые могут быть оттранслированы LISP-компилятором через Интернет.

Система ELM-ART II была разработана для перевода обычных учебников в электронные. Она улучшает представление знаний, имевшееся в ELM-ART. Понятийная сеть иерархически организована в лекции, секции, подсекции и конечные страницы. Каждый элемент понятийной сети имеет слот, содержащий текст для страницы и информацию о связи этой страницы с другими элементами. Статические слоты содержат предварительные требования, родственные понятия и результат. Конечные страницы содержат тестовые слоты. Страницы с задачами имеют определенный слот для хранения описания программной задачи. Индивидуальная модель ученика хранит посещенные страницы, решенные тесты и решенные программные задачи, помечая соответствующие понятия в понятийной модели как «известные».

Прямое руководство осуществляется с помощью кнопки «следующий лучший», подсказка предоставляется с помощью нахождения наиболее подходящего примера из индивидуальной истории обучения, основываясь на диагностике программного кода, использованного учеником в своих решениях.

Системы способны делать заключения о знаниях пользователей, основываясь на помеченных понятиях в понятийной модели. Все предварительные требования, считающиеся необходимыми для известных пользователю понятий, также рекурсивно помечаются как известные.

Система INTERBOOK [72] поддерживает создание электронных учебников на базе иерархически структурированных файлов MS-Word. Для этого должны быть выполнены такие операции, как создание списка понятий предметной области, структурирование и аннотирование страниц с выдаваемыми результатами и предварительными требованиями, трансляция в HTML и разбор информации на структуры LISP.

INTERBOOK использует и модель предметной области, и модель пользователя. Глоссарий и учебник основаны на модели области. Глоссарий есть визуализированная понятийная сеть. Структура глоссария напоминает дидактическую структуру знаний данной области. Каждый пункт глоссария соответствует одному из понятий области. Вдобавок к этому каждая запись в глоссарии содержит ссылки на все разделы книги, которые используют это понятие.

Каждый элемент учебника помечен некоторыми понятиями модели области. Эти понятия играют различную роль. Некоторые из них описывают *результаты* — знания, которые пользователь получил после изучения страницы, другие описывают *предварительные* условия, или знания, необходимые для понимания этой страницы.

INTERBOOK поддерживает адаптивную аннотацию ссылок, используя метафору светофора. Подсказка, основанная на предварительных условиях, осуществляется с помощью предоставления аннотированного списка страниц, содержащих предварительно необходимую информацию.

Упорядочение страниц производится в три этапа. Вначале система вычисляет общее количество «очков», отражающих предполагаемое состояние знаний, для каждого понятия. Основываясь на этих цифрах, система решает, хорошо усвоено понятие или нет. Затем система решает, какие страницы содержат рекомендованную обучающую информацию либо недостающие предварительные знания. Ссылки на понятия и разделы разных образовательных состояний аннотированы различными пиктограммами. Наконец, система выбирает наиболее оптимальную страницу среди всех доступных страниц, на которых вводятся неизученные понятия и все предварительные условия которых уже известны. Всем страницам присваивается определенный приоритет их представления, который выводится из базового значения в соответствии с состоянием знаний о требуемых и вводимых понятиях.

Подход к реализации адаптивного пользовательского интерфейса для проекта INTERBOOK был предложен в работе [71]. Модель области и модель ученика используются для его представления. Каждое свойство интерфейса рассматривается как понятие области, а каждая подсказка — это обучающий элемент. Применяя алгоритм упорядочения страниц, подобный описанному выше, генерируются последовательность свойств интерфейса, требующих изучения, и последовательность подсказок, которые нужно показать пользователю.

Система PT (персонализированная текстовая система) — это учебник по изучению программирования на языке C [117, 118]. Она использует обычную книгу о языке C и генерирует на ее основе гипермедиа-систему. Курсом, который поддерживается PT, является курс по программированию на языке C для программистов на языке Pascal.

PT использует стереотипную пользовательскую модель целевой аудитории (программистов на языке Pascal) в дополнение к индивидуальной модели. Стереотип дает определенные значения компонент знания, инициализируя таким образом модель пользователя. В индивидуальной модели хранятся значения знаний индивидуального пользователя во время его работы с PT.

Для того чтобы сделать возможной адаптацию, PT использует сходство языков Pascal и C при представлении информации пользователю. Команды препроцессора добавляются в «сырой» HTML-код страницы из модели пользователя, например:

#define PASCAL 3,

#define active-learner 1,

#if PASCAL > 2.

Команды «if» препроцессора на HTML-странице используются для контроля того, какие именно части страницы передаются пользователю. Эта же техника препроцессирования используется для выбора ссылок.

Система PUSH [106] (подсказка, настроенная на пользователя и план) ставит целью разработку и тестирование интеллектуальных справочных решений к задачам поиска информации. Пользователь может вводить вопросы, в том числе повторные, или перемещаться по графическому представлению предметной области. Когда пользователь сформулировал вопрос, он может воздействовать на ответ системы, открывая и закрывая подсекции, изменяя графику или выбирая последующие предлагаемые вопросы.

Знания о предметной области моделируются isA-иерархией. Идея системы состоит в том, чтобы заготовить вопросы, которые могут возникнуть у пользователя при чтении документа. Вопросы затрагивают родственные понятия и знания. Так как комбинаторная конструкция всех возможных

вопросов, основанных на понятиях и их взаимосвязях в isA-иерархии, будет слишком сложной и содержащей слишком много информации, множество возможных вопросов ограничено понятиями, содержащимися на странице, и повторными вопросами для этих понятий. Система реализует подход к запросам, основанный на правилах. Как только пользователь верифицирует каждую из своих задач, выбирая некоторые из предлагаемых ссылок, система получает практически точную информацию о знаниях пользователя.

Система AHA поддерживает бесплатный Web-курс по *гипермедиа-структурам и системам* [74, 86]. AHA (адаптивная гипермедиа-архитектура) может быть использована для генерации условных тестов, а также для адаптации ссылочной структуры с помощью удаления, сокрытия и аннотации ссылок. Команды препроцессора на HTML-страницах используются CGI-скриптами для фильтрации содержания фрагментов страницы, таким способом реализуется адаптация на уровне содержания. Та же техника препроцессирования используется для адаптации на уровне ссылок.

AHA! версии 1.0 [92] делает возможным создание сайтов, в которых ссылки могут быть спрятаны или аннотированы, а фрагменты — включены или опущены, основываясь на правилах адаптации и модели пользователя с очень гибкой структурой (продукционные правила работают в структуре концептов и атрибутов). AHA! 1.0 — инструмент широкого назначения в том смысле, что он не навязывает один стиль презентации и что адаптация может быть основана на произвольных событиях (arbitrary events) и зависимостях между концептами (в отличие от, например, обучающих систем, в которых предполагается монотонный процесс усвоения знаний посредством чтения страницы). Однако в данной версии AHA! все еще довольно простая система. Она работает только с правилами, созданными вручную, и условное включение фрагментов — единственный тип адаптации содержания, который она поддерживает.

В AHA! версии 2.0 [87, 88] авторы направили силы на создание более богатой структуры модели пользователя, предметной области и модели адаптации с более развитыми *требованиями* и правилами *генерации*. Новые особенности системы, разработанные авторами, основаны на ссылочной (reference) модели AHAM [91], которая является расширением хорошо известной ссылочной модели Dexter для гипермедиа [103]. AHA! отличается от ссылочной модели AHAM [91] тем, что в ней не отделена *модель предметной области* от *модели адаптации*. В AHA! автор определяет *концепты* наряду с *требованиями*, определяющими условия, когда пользователь «готов» перейти к концепту, и *правилами генерации*, которые определяют, как поведение просмотра (browsing behavior) пользователя интерпретирует-

ся при обновлении модели пользователя. *Правила генерации* являются *продукционными правилами* (condition-action rules), как принято в теории активных баз данных [60]. AHA! 2.0 поддерживает хранение структуры концептов в XML-файлах или в базе данных mySQL.

Система OLAE [124] нацелена на дифференцированное и надежное определение знаний обучающегося в области физики. Сеть Байеса [132] строится для каждой проблемы, над которой работает пользователь. Эта сеть отражает, среди прочего, вероятность того, что пользователь вводит определенные уравнения, если он знает соответствующие правила. Таким образом, система использует ретроспективную диагностику, называемую *отслеживанием знаний*. Система POLA [78] разработана для *отслеживания моделей*: она может вызываться многократно во время процесса решения проблемы. POLA строит базовую сеть Байеса с помощью приращения узлов, добавляя их каждый раз, когда обучающийся производит наблюдаемое действие.

Система POKS [95] основана на когнитивной теории структур знания. Она строит сеть выводов на элементах знаний из небольшого образца из множеств данных пользователя и использует эту «наведенную» сеть для доступа к состоянию знаний, обладая ограниченным количеством наблюдений или вопросов. Система применяется для адаптированного построения пользовательских интерфейсов. Представлено моделирование перекрытий.

Система EPI-UMOD [137] реализует модель стереотипного пользователя, основанную на сетях Байеса. Она использует отдельную сеть Байеса для каждого стереотипа, в которой реализованы специальные условные зависимости между единицами знаний. Каждый атрибут этого стереотипа представляет собой утверждение, что пользователю известно определенное понятие.

Система HYDRIVE [125] нацелена на обучение технического и летного персонала в области функционирования гидравлических систем самолета F15. Основной упор делается на понимание системы и стратегии разрешения проблем, а не на оптимизацию действий, которые необходимо предпринимать в заданном пункте проблемы. Авторы используют иерархическую модель необходимых способностей обучающихся и строят сеть Байеса для всего сценария приложения. Однако эта сеть модифицируется только частично.

Система KBS-гиперкниг [104, 105] ставит своей целью построение структуры для разработки и поддержки открытых адаптивных гипермедиасистем в Интернете. Она реализует обучение, основанное на проектах [141].

В системах баз знаний (KBS, knowledge-based systems) понятия связаны друг с другом на базе понятийной модели гиперкниги. Наблюдения за пользователями производятся тогда, когда они выполняют некоторые проекты в библиотеке проектов гиперкниги. Каждый элемент гиперкниги индексирован некоторыми понятиями из области знаний. Строится отдельная модель знаний, содержащая понятия знаний из предметной области и их обучающие зависимости. Таким образом, сам по себе гипертекстовый документ не содержит никакой информации о предварительных требованиях или результатах.

KBS использует метафору светофора для адаптивной аннотации. Алгоритм упорядочения страниц генерирует последовательность чтения в соответствии с целями и знаниями пользователя. Чтобы помочь пользователю проложить собственный путь через гиперкнигу, система также генерирует следующий обучающий шаг путем сравнения текущего состояния знаний пользователя с состоянием знаний, которое он должен иметь после завершения работы с книгой.

KBS поддерживает обучение, основанное на целях. Пользователи могут определять собственные цели обучения или запрашивать следующую цель у системы. Для каждой из этих целей генерируется последовательность чтения, содержащая необходимые знания (как предварительные требования, так и текущее состояние знаний) для достижения цели. Вдобавок к этому, выбираются подходящие проекты и предлагается информационный индекс, содержащий как документы из самой гиперкниги, так и другие доступные в Интернете материалы.

Выбор подходящих проектов основывается на алгоритмах, которые отражают как предварительные знания, необходимые для выполнения проекта, так и то, насколько хорошо проект соответствует текущей цели обучения.

KBS также адаптируется к различным скоростям обучения пользователей, поддерживая эту разновидность обучения, основанного на целях. Пользователи могут сами определять, сколько и что именно они хотят изучить на следующем шаге. Если система замечает, что обучаемый овладел проектами повышенной сложности достаточно успешно, она модифицирует свою оценку в отношении пройденных им тем, а также предварительных знаний пользователя. Таким образом, обучаемый сможет перейти к другим, более продвинутым темам. Если система обнаруживает, что пользователь недостаточно знаком с некоторой темой, она предлагает сходные примеры или проекты, содержащие минимальное количество новой информации.

Техника реализации, использованная в KBS, представляет собой сеть Байеса над полной моделью предметной области. Когда бы ни были произ-

ведены наблюдения над конкретным пользователем, они далее распространяются по всей сети.

Важное внимание в KBS уделяется расширяемости системы по отношению к Интернету. Чтобы иметь возможность создавать *открытые* адаптивные гипермедиа-системы, выбранный в KBS подход к индексированию позволяет обрабатывать каждую информационную единицу одинаково независимо от ее происхождения. Таким образом, HTML-страницы, найденные в Интернете, могут быть интегрированы в систему таким же образом, как и документы, размещенные в библиотеке гиперкниги.

Система TILE [119] — интегрированная адаптивная система для дистанционного обучения с моделью пользователя, основанной на клиент-серверном подходе. TILE поддерживает модели отдельных студентов и групп студентов.

Индивидуальная модель содержит четыре основных составляющих: глобальные предпочтения; специфическое представление содержания, связанное с предпочтениями обучаемого; компетентность в предметной области; история работы студента. Она используется системой для обеспечения адаптивного навигационного управления, выбора гранулированности содержания предметной области, обеспечения основанных на контекстах экскурсий к другим единицам обучения, нахождения аналогий с ранее изученным материалом, создания прямых ссылок на ранее изученный материал, обеспечения динамических сообщений и обратных связей.

Система суммирует общее поведение студентов и предпочтений для создания и сопровождения групповой модели. Студенты, использующие систему, разбиваются на различные группы, соответствующие их поведению и компетентности в предметной области, которые отражаются в модели определенных студенческих групп. Структура групп многомерна, и один студент может принадлежать одной или нескольким таким группам.

Групповая и частично индивидуальная модели размещаются на сервере, в то время как у клиента находится только частично индивидуальная модель.

Функционально система TILE содержит следующие компоненты: модуль студенческой модели; модуль предметной области (представление базы знаний); модуль эксперта, поддерживающий единицы обучения (лекции, примеры, тесты и т.д.), каталог ошибок и объяснения для элементов учебного плана; модуль обучения, содержащий действия по адаптации и отслеживания студенческих действий.

Система SAC [77] — система регулируемого адаптивного обеспечения курсов, основанная на ссылочной модели AHAM [91]. Структура курса, материалы и цели обучения представляются абстракциями вершин курса,

единиц курса и материала курса, как это описано в модульной системе обучения MTS [146].

Курсовое обеспечение реализуется с помощью архитектуры модели трехярусного приложения. Первый ярус реализуется с помощью браузера, связанного с представлением обеспечения курса. Средний уровень реализуется и обеспечивается комбинацией Web-сервера и сервера приложений. Сервер приложений генерирует XML, DHTML и клиентский Джава-скрипт, динамически используя модельную информацию, сохраняемую в третьем ярусе системы базы данных.

Адаптивные возможности SAC обеспечиваются адаптивным агентом, который состоит из следующих трех частей: предвходовый адаптивный механизм, онлайновый индивидуальный адаптивный механизм и адаптивный механизм предметной области.

Система WebVT [145] является адаптивной сетевой интеллектуальной программой обучения языку с помощью компьютера, предназначенной для пассивного овладения английским языком лицами, для которых он не является родным языком.

Система объединяет возможности интеллектуальных систем обучения и адаптивных гипермедиа-систем. Она использует комбинацию стереотипного моделирования и моделирования перекрытий для инициализации студенческой модели. Эта модель в дальнейшем уточняется в процессе наблюдения за студентом, когда он работает с системой. Результирующая модель студента используется для аннотаций ссылок на топики, представляемые студенту. Кроме того, она также используется в процессе диагностики ошибок и адаптации обратной связи и советов, выдаваемых студенту.

Система ITS (интеллектуальная система обучения) [134] предназначена для обучения использованию новых технологий преподавателей высших учебных заведений. Она предлагает единицы курса, покрывающие потребности пользователей с различными уровнями знаний и различными характеристиками.

Система настраивает представление учебного материала на различные пользовательские потребности с использованием технологий искусственного интеллекта для спецификации каждой пользовательской модели и принятия педагогических решений. Это достигается с помощью экспертной системы, которая использует формализм гибридного представления знаний, интегрирующий системы продукций с нейровычислениями.

Выводы по главе 2

В главе мы рассмотрели методы и средства адаптации, используемые в современных обучающих Web-системах. Не вызывает сомнения, что как адаптивные, так и интеллектуальные технологии могут повысить качество образовательных систем, используемых в сети. Например, адаптивное представление может сделать представление учебного материала более удобным для использования. Адаптивная поддержка в навигации и адаптивное построение последовательности могут использоваться как для повсеместного контроля над курсом обучения, так и для помощи студенту в выборе наиболее подходящих тестов и заданий. Поддержка в решении задач и интеллектуальный анализ решений могут значительно улучшить процесс выполнения заданий студентами, обеспечивая их интерактивной и интеллектуальной обратной связью на фоне значительного уменьшения нагрузки на преподавателя. Технологии подбора моделей могут улучшить как управление дистанционными курсами, так и взаимодействие между обучаемыми и преподавателями.

Поэтому перспективными представляются такие сетевые обучающие системы, которые сводят воедино идеи гипермедиа-систем и интеллектуальных обучающих систем и делают возможным персонализированный доступ к информации и интеллектуальную поддержку обучения. Эти свойства стали особенно важны для Web-систем дистанционного обучения с тех пор, как обучаемые стали учиться в основном самостоятельно (обычно из дома). Интеллектуальное и личное содействие, которое могут дать учитель или студент-сокурсник при обычном (аудиторном) обучении, при дистанционном обучении нелегко достижимо. Адаптивность важна для программного обеспечения дистанционного обучения еще и потому, что оно должно использоваться намного более разнообразным множеством студентов, чем любое «однопользовательское» учебное приложение. Сетевое программное обеспечение, разработанное для одного класса пользователей (с определенным складом ума), может совсем не подойти другим обучаемым.

К недостаткам современных адаптивных сетевых систем обучения следует отнести то, что они слабо поддерживают проблемный подход к обучению, а также то, что они в большинстве случаев ограничиваются использованием информации о посещении страниц пользователем для обновления его знаний. Как правило, системы фиксируют факт чтения пользователем определенной информации и на этой основе обновляют свою оценку его знаний. Иногда они учитывают время чтения или последовательность прочитанных страниц для углубления этой оценки. Хотя это оправданный под-

ход, его недостатком является трудность измерения знания, которое пользователь приобретает во время «чтения» страницы.

Кроме того, несмотря на имеющий потенциал, адаптивные и интеллектуальные технологии пока еще не нашли своё место в «реальной» виртуальной аудитории, т.е. пока их нет в массово используемых средствах обучения. Большинство систем, обсуждавшихся выше, — это типичные «лабораторные» (экспериментальные) системы, которые вообще не имели практики использования в реальных дистанционных курсах. Небольшая часть систем (в основном это системы из семейств ELM-ART и AHA) имеют небольшую практику использования, скорее экспериментального характера. В то же время ни одна из существующих коммерческих и «университетских» систем обучения, поддерживающих сотни дистанционных курсов, не использует ни адаптивные, ни интеллектуальные технологии.

Здесь автор разделяет позицию П. Бруселовского [63], заключающуюся в следующем. Сетевое образование само по себе относительно молодо. До настоящего времени различные компании, производящие системы для сетевого образования, были способны конкурировать на рынке, создавая простые неадаптивные системы. Однако большое количество экспериментальных систем демонстрирует явные преимущества адаптивных и интеллектуальных технологий. По ходу увеличения конкуренции на рынке образовательных Web-систем «быть адаптивной» или «быть интеллектуальной» станет важным фактором для завоевания покупателей. Традиционные компании создания систем сетевого образования начнут использовать адаптивные и интеллектуальные методы. Команды исследователей с серьезным опытом использования адаптивных и интеллектуальных технологий получат возможность для вывода своих технологий на рынок. Первыми технологиями для использования в коммерческих системах, вероятно, будут технологии построения последовательностей (последовательность страниц и последовательность вопросов), так как они вполне подходят к существующей структуре обучающих систем дистанционного образования. Потом наступит очередь адаптивной поддержки в навигации и подбора модели. Технологии поддержки в решении задач будут оставаться на уровне исследований дольше, хотя мы можем ожидать появления небольших Web-тренажеров, предназначенных для поддержки изучения тех или иных частей некоторых предметов. Можно надеяться, что в ближайшие годы появится несколько примеров адаптивных и интеллектуальных систем коммерческого уровня так же, как и множество новых и перспективных разработок экспериментальных систем для дистанционного обучения.

ГЛАВА 3. АДАПТИВНАЯ СИСТЕМА WAPE ДЛЯ ПОДДЕРЖКИ ДИСТАНЦИОННОГО ОБУЧЕНИЯ ПРОГРАММИРОВАНИЮ

Глава посвящена проекту WAPE, работа над которым ведется в Институте систем информатики СО РАН [29–36, 114, 115]. Цель проекта — разработка адаптивной среды дистанционного обучения WAPE, поддерживающей активное индивидуальное обучение программированию в рамках проблемного подхода и соединяющей возможности адаптивных гипермедиа-систем и интеллектуальных обучающих систем.

Глава начинается с общего описания системы WAPE и ее возможностей с точки зрения обучаемых (студентов). Возможностям, которые система предоставляет другим пользователям (администраторам, лекторам и инструкторам), посвящен разд. 3.2. В разд. 3.3 рассматривается модель знания курса, а в разд. 3.4 и 3.5 — вопросы моделирования знаний студента и использования при этом моделировании механизма сетей Байеса. Моделям тестирования и видам тестов посвящен разд. 3.6. В разд. 3.7 и 3.8 рассматриваются проекты (упражнения и задания) подсистемы CLASS, играющей роль виртуальной семинарской аудитории в рамках системы WAPE. Разд. 3.9 содержит описание подсистемы PRACTICE, предназначенной для использования при прохождении студентами компьютерного практикума по курсу, поддерживаемому системой WAPE. Виды аннотирования проектов и другие действия, которые осуществляет система в помощь студентам, рассматриваются в разд. 3.10. Разд. 3.11 посвящен режимам работы студентов, а вопросы обновления модели знаний студента и развития курса описаны в разд. 3.12 и 3.13. В разд. 3.14 описываются работы по реализации системы WAPE. Завершают изложение выводы по главе 3.

3.1. Система WAPE

Система WAPE ориентирована на поддержку дистанционного обучения и предполагает четыре типа пользователей: студенты, инструкторы, лекторы и администраторы. Все пользователи осуществляют доступ к системе через стандартный Web-браузер, который представляет HTML-документы, предоставляемые HTML-сервером на стороне сервера.

После авторизации пользователя в качестве студента открывается подходящее меню команд.

Система WAPE поддерживает три уровня процесса обучения:

- студент изучает теоретический материал в некоторой специфической области с использованием гипертекстовых учебников и задачников;

- система тестирует концептуальные знания студента, соответствующие изученному теоретическому материалу;
- студент под управлением системы выполняет учебные проекты, решая задания и упражнения.

Третий уровень рассматривается как основной в использовании системы WAPE; для того чтобы изучить курс, поддерживаемый системой WAPE, студент должен справиться с набором *проектов* (заданий и упражнений), который инструктор подбирает студенту строго индивидуально.

Другой тип задач, поддерживаемый системой WAPE, — это так называемые *тесты*. В отличие от проектов, решение о выполнении (или невыполнении) которых принимается инструктором, тесты — это вопросы, правильность ответов студентов на которые система оценивает полностью автоматически.

Ориентация на цели обучения является одним из важных свойств нашей среды WAPE. Поскольку мы не хотим фиксировать путь обучения студента или студенческой группы от начала до конца, студенты свободны в определении своих собственных целей обучения и своих собственных последовательностей обучения. На каждом шаге они могут обращаться за помощью к системе, запрашивая подходящий материал, последовательности обучения и советы по примерам и проектам. Если студенту необходим совет по нахождению своего собственного пути обучения, он может спросить систему о следующей подходящей цели обучения.

Система WAPE предназначена для обслуживания многих студентов с различными целями, знанием и опытом. В нашей системе основной упор делается на знание студентов, уровень которого может весьма сильно варьироваться у разных студентов. Более того, состояние знаний студента изменяется в процессе работы с системой. Поэтому большое внимание уделяется возможностям адаптивности в нашем проекте.

Система WAPE предоставляет лектору и инструкторам средства для управления мониторингом взаимодействия студентов с системой. Возможно определять те действия студента, которые нуждаются в реакции со стороны преподавателя. Например, когда студент завершает выполнение задания (или упражнения), сообщения посылаются инструктору, отвечающему за мониторинг работы данного студента.

Открытые дискуссии, поддерживаемые системой WAPE, обеспечивают полную виртуальную атмосферу телекласса, включая возможности кооперативного изучения курса вместе с другими студентами и средства кооперативного преподавания для инструкторов и лекторов.

3.2. Возможности администраторов и преподавателей

Помимо студентов, система WAPE поддерживает три типа пользователей: инструкторов, лекторов и администраторов. Эти типы пользователей различаются как по своим правам, так и возможностям работы с системой. Каждому типу пользователей соответствует свой интерфейс, поддерживаемый системой.

Интерфейс администратора поддерживает ряд административных функций организации учебного процесса, которые разбиваются на следующие две части.

1. Управление курсами и преподавателями. В этой части можно осуществлять создание и удаление курсов и преподавателей, а также связывать преподавателей с курсами, потоками и группами в качестве лекторов и инструкторов и заменять одного лектора курса или инструктора учебной группы на другого.
2. Управление студентами. В этой части можно создавать и удалять потоки, группы и отдельных студентов, а также переводить студентов из одной группы в другую.

Интерфейс лектора поддерживает следующие основные функции.

1. Редактирование учебной информации курса. Сюда входят возможности по включению новых учебников в курс, пополнению гиперкниг курса новыми примерами, созданию или улучшению примеров проектов, по пополнению заданий новыми тестами и эталонными решениями, а также пополнению пространств тестов новыми тестами.
2. Общение со студентами и инструкторами. Данная возможность реализована в виде общих и преподавательских форумов, администратором которых является лектор. В общих форумах могут участвовать как студенты, так и преподаватели, а в преподавательских только лектор и инструкторы. После включения лектором некоторой общей темы для обсуждения любой студент и любой инструктор могут написать свое мнение по обсуждаемому вопросу, но у лектора есть возможность удалять и редактировать любые сообщения.
3. Просмотр статистики. Практически любые действия студента и инструктора заносятся в таблицу статистики и могут быть рассмотрены лектором. В частности, лектор может посмотреть, как часто и сколько времени студенты его потока тратят на обучение, сколько раз и какие тесты они проходили (с фиксацией времени и успеха

прохождения). Здесь же он может узнать текущее состояние модели знаний каждого студента, а также какие задачи были им уже решены, а какие нет. Для каждой отдельной задачи можно также узнать количество раз, которое студент пытался ее решить, и посмотреть все варианты решений, которые студент предложил, вместе с комментариями инструктора.

4. Управление мониторингом. Система предоставляет лектору возможности управления мониторингом взаимодействия студентов и инструкторов с системой. Здесь он может определить те действия студентов и инструкторов, которые нуждаются в его реакции. Каждый раз, когда выбранные действия будут происходить, лектор будет получать соответствующие сообщения.

Интерфейс инструктора поддерживает следующие основные функции.

1. Проверка и оценка заданий студентов. Каждое задание, выполненное студентом, должна быть проверено и оценено инструктором. После того как студент решает задание и проверяет его на имеющихся тестах, он отправляет решение на проверку инструктору. Для проверки правильности и оценки качества студенческого решения инструктор может использовать эталонные решения данного задания, если они есть. При этом он должен не только оценить данное решение (отклонить или принять с некоторой положительной оценкой), но и написать комментарий с разъяснением причин такой оценки.
2. Общение со студентами и преподавателями. Указанные возможности образуют общую и приватную части. Общая часть реализована в виде форумов, администратором которых является лектор, а также форумов, которые администрирует инструктор и создает их для своих студентов. Приватная часть дает инструктору возможность приватного общения с любым своим студентом. Она выполнена в виде гостевой книги, т.е. выводится просто последовательный список сообщений в порядке, обратном к порядку их написания (т.е. последнее сообщение всегда выводится первым). Преподаватель может отвечать на сообщения студентов, и наоборот. Ответы выводятся под соответствующими сообщениями и выделяются другим цветом и шрифтом.
3. Просмотр статистики. Интерфейс просмотра статистики у инструктора имеет те же самые возможности, что и у лектора. Различие лишь в том, что инструктор может просматривать только свою статистику и статистику студентов своих групп.

4. Управление режимом работы и мониторингом. Система предоставляет инструктору возможности управления режимом работы каждого своего студента, а также мониторингом взаимодействия этого студента с системой. Здесь он может определить, как знания студента будут влиять на его возможность решать проекты, а также выбрать те действия студента, которые нуждаются в его реакции. Каждый раз, когда студент будет совершать выбранные инструктором действия, соответствующие сообщения будут посылаться инструктору. Например, когда студент завершает выполнение задания (или упражнения), сообщения всегда посылаются инструктору, отвечающему за группу, в которой работает данный студент.

3.3. Модель знаний курса

Центральным объектом модели обучения в WAPE является некоторый учебный курс (или просто курс), который представляет реальный курс, читаемый в некотором университете, реальном или виртуальном.

Каждый курс имеет своего лектора, который создает и поддерживает глоссарий, проекты, тесты, учебники и задачники по курсу (в качестве учебного материала), а также курирует проблемное обучение групп студентов, объединенных в студенческий курс (или поток), осуществляемое под руководством группы инструкторов (ассистентов лектора). Информация о завершении обучения одного потока студентов сохраняется в системе и может использоваться лектором для совершенствования его курса до того, как будет набран новый поток студентов для обучения.

В основе курса лежит его *модель знаний* (S, U, W), которая состоит из конечного непустого множества *единиц знаний* S и двух бинарных отношений U и W на S, т.е. $U \subseteq S \times S$ и $W \subseteq S \times S$, удовлетворяющих следующим свойствам для любых $p, q \in S$:

1) $(p, q) \in U$ тогда и только тогда, когда p является составной единицей знания, которая является объемлющей для q;

2) $(p, q) \in W$ тогда и только тогда, когда единица p должна быть изучена до изучения q.

Предполагается, что в модели знаний (S, U, W) пара (S, U) образует лес, а (S, W) является ациклическим орграфом.

На основе модели знаний курса строится *глоссарий* курса, в котором каждая единица знаний представлена одним (или несколькими) ключевым словом (или фразой) и дополнена множеством ссылок на те элементарные информационные ресурсы курса (элементарные информационные ресурсы

учебников и задачников, а также примеры, тесты и проекты), содержание которых относится к данной единице знаний.

Каждый учебник или задачник имеет вид гипертекстовой книги (*гиперкниги*), обладающей иерархической структурой: книга, глава, параграф.

В общем случае гиперкнига содержит последовательность глав, каждая из которых может в свою очередь состоять из последовательности параграфов.

Элементарный информационный ресурс гиперкниги — это либо параграф, либо глава, не содержащая параграфов.

Особую группу элементарных информационных ресурсов гиперкниги образуют *примеры*. Они не содержат теоретического материала курса и служат для его пояснения.

Каждый элементарный информационный ресурс курса представлен отдельной Web-страницей и индексируется некоторым набором единиц знаний, описывающих содержание этого ресурса. Происхождение информационного ресурса несущественно для индексирования, только содержание определяет его информационный индекс.

Пусть $S \neq \varnothing$ — множество всех единиц знаний, $P(S)$ — множество всех подмножеств множества S, а H — множество всех информационных ресурсов, тогда *карта содержания* — это функция

$I : H \to P(S) \setminus \varnothing$,

которая каждому информационному ресурсу $h \in H$ сопоставляет *информационный индекс* этого ресурса $I(h)$ — непустое множество всех тех единиц знаний, которые связаны с данным информационным ресурсом: если h является тестом или проектом, то $I(h)$ состоит из тех единиц знания, которые используются в h, а если h является информационным элементом гиперкниги, то $I(h)$ состоит из тех единиц знания, объяснение которых в той или иной степени содержится в данном информационном ресурсе h.

3.4. Моделирование знаний студента

Знания студента x моделируются *вектором знаний*
$K(x) = (p_1, \ldots p_n)$, где

- $n = |S|$ — число единиц знаний в модели знаний курса, а
- для любого i, $1 \le i \le n$, $p_i = p(s_i / E_i)$ — условная вероятность, описывающая предположение системы о том, что студент x обладает знанием единицы знания $s_i \in S = \{s_1, s_2, \ldots, s_n\}$ на базе тех свидетельств E_i, которые система собрала о знании студентом x единицы знания s_i в процессе мониторинга его работы над курсом.

Каждый элемент p_i вектора знаний $K(x)$ выражает степень знания единицы знания s_i, которым обладает студент x в данный момент.

Мы используем четыре степени такого знания:

- знания эксперта в области данной единицы знания — обозначаются E (отличные знания);
- знания продвинутого пользователя — обозначаются F (несколько затруднительных моментов в понимании s_i, но в целом понятие вполне усвоено);
- знания начинающего — обозначаются A (много затруднительных моментов, понятие s_i плохо усвоено);
- знания новичка — обозначаются N (пользователь еще не готов для работы с данным понятием, оно им не усвоено).

Таким образом, единицы знания являются с одной стороны понятиями, описывающими предметную область курса, а с другой — случайными переменными с четырьмя дискретными значениями E, F, A и N, кодирующими степень знаний данного студента.

Свидетельства, получаемые системой в процессе мониторинга действий студента, изменяются со временем. В типичном случае знание студента возрастает во время работы с курсом, хотя недостаток знания также воспринимается системой как свидетельство. Поскольку каждый отслеживаемый результат производимого наблюдения за студентом сразу же заносится в свидетельства, вектор знаний в любой момент дает срез (моментальный снимок) текущего уровня знаний студента.

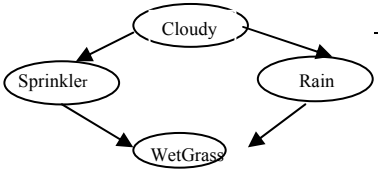
Обновление свидетельств о студенте происходит только при выполнении им тестов и проектов. При этом никакое тестирование не позволяет студенту получить от системы оценки “отличные знания”, и он может достичь уровня знаний эксперта только за счет успешного выполнения проектов.

Пусть $P = (p_1, \ldots p_n)$ и $Q = (q_1, \ldots q_n)$ — два произвольных вектора знаний, тогда

- $P \geq Q$, если $p_i \geq q_i$ для любого I;
- $min\ (P, Q)$ — это такой вектор знаний $(w_1, \ldots w_n)$, что $w_i = min(p_i, q_i)$ для любого I;
- $max\ (P, Q)$ — это такой вектор знаний $(w_1, \ldots w_n)$, что $w_i = max(p_i, q_i)$ для любого i.

$P(C = F)$	$P(C = T)$
0.5	0.5

C	P(S=F)	P(S=T)
F	0.5	0.5
T	0.9	0.1



C	P(R=F)	P(R=T)
F	0.8	0.2
T	0.2	0.8

S	R	P(W=F)	P(W=T)
F	F	1.0	0.0
T	F	0.1	0.9
F	T	0.1	0.9
T	T	0.01	0.99

Рис. 3.1. Простая сеть Байеса

3.5. Механизм сетей Байеса: вычисление вероятностей

Сети Байеса являются мощным инструментом вывода в графах с зависимыми случайными вершинами [58, 79, 80, 132, 138]. Мы используем такую сеть для вычисления распределения вероятностей для каждой единицы знаний и, следовательно, для вычисления векторов знаний пользователей.

Сеть Байеса представляет собой ациклический орграф со следующими свойствами:

- каждая вершина графа представляет случайную переменную;
- имеется дуга из X в $Y \neq X$ в каждом случае, когда Y зависит от X;
- каждая вершина помечена таблицей условной вероятности, которая определяет воздействие на вершину ее непосредственных предшественников.

На Рис. 3.1. приведен пример простой сети Байеса, адаптированный из [138]. Вершины сети представляют случайные логические переменные, принимающие значения: T (истина) или F (ложь).

Чтобы построить сеть Байеса, которая вычисляет распределение вероятностей для каждой единицы знаний курса для конкретного студента, требуется выполнить два действия. Во-первых, сгенерировать некоторый ациклический граф, имеющий единицы знаний в качестве вершин и обучающие зависимости между ними в качестве дуг, и, во-вторых, определить таблицы вероятности для всех вершин.

Для наших целей мы строим сеть Байеса со случайными переменными, которые дают распределение вероятностей для вычисления знаний пользователя. Мы используем в качестве вершин случайные переменные с четырьмя дискретными значениями из множества $\{E, F, A, N\}$ и проводим дугу из вершины X в вершину Y в том и только том случае, когда $Y < X$ и не существует такого Z, что $Y < Z < X$, где $Y < X$, если Y зависит от X. т.е. если $(X, Y) \in W$ или $(Y, X) \in U$.

Известно, что точный вывод в сетях Байеса является *NP*-трудной проблемой [79]. Вместе с тем существуют линейные по времени алгоритмы для тех сетей Байеса, базовые[2] графы которых не содержат циклов, а также можно привести несколько преобразований, направленных на удаление из сети Байеса не полностью ориентированных циклов[3]: кластеризация, кондиционирование и стохастическая симуляция.

Алгоритмы кластеризации «склеивают» две или более вершины в одну для устранения не полностью ориентированных циклов (см., например, [93, 132]). Пример работы алгоритма кластеризации показан на Рис. 3.2. Здесь переменные B и C объединяются в единую вершину BC. Область значений BC является декартовым произведением областей значений B и C. Например, если осуществить кластеризацию сети Рис. 3.1, где B=Sprinkler и C=Rain, для вершины кластера BC получаем таблицу условных вероятностей, изображенную на Рис. 3.3.

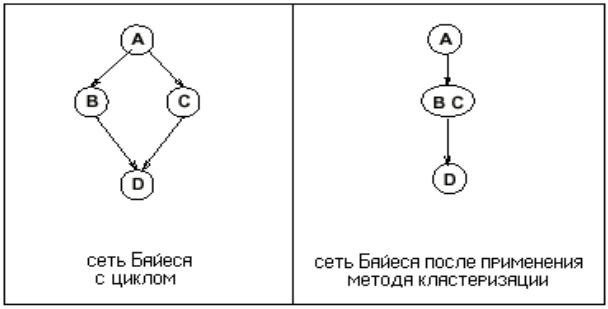


Рис. 3.2. Пример удаления не полностью ориентированного цикла с помощью кластеризации

[2] Базовый граф сети Байеса — это неориентированный граф, получаемый из данной сети снятием ориентации ее дуг.

[3] Подграф сети Байеса, соответствующий циклу ее базового графа, называется не полностью ориентированным циклом сети.

C	P(SR=TT)	P(SR=TF)	P(SR=FT)	P(SR=FF)
F	0,10	0,40	0,10	0,40
T	0,08	0,02	0,72	0,18

Рис.3.3. Таблица вершины-кластера

Нахождение подходящей вершины-кластера является важной частью алгоритма кластеризации. Поскольку таблицы условной вероятности вершины-кластера содержат декартовое произведение областей значений «склеиваемых» переменных, количество вычислений для сети может расти экспоненциально [138].

Методы кондиционирования удаляют из сети Байеса не полностью ориентированные циклы, преобразуя одну исходную сеть в несколько сетей более простой структуры (см., например, [94]). Каждая из этих сетей содержит одну или более случайных переменных, конкретизированных одним из их значений. К примеру, на Рис. 3.4 изображена сеть Байеса, в которой A — случайная переменная с четырьмя дискретными значениями, и применение метода кондиционирования приводит к ее замене на четыре сети, в каждой из которых переменная A конкретизирована одним из ее четырех значений.

Поскольку количество сетей в результате кондиционирования растет экспоненциально относительно количества конкретизированных переменных и их значений, этот подход не всегда применим, поскольку может потребовать поддержания большого количества построенных таким образом сетей Байеса.

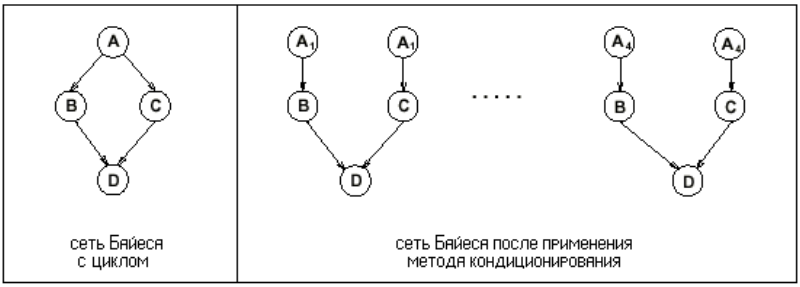


Рис. 3.4. Пример удаления не полностью ориентированного цикла с помощью кондиционирования

Методы стохастической симуляции выполняют многократную симуляцию сети, вычисляя аппроксимации точных значений (см., например, [132]).

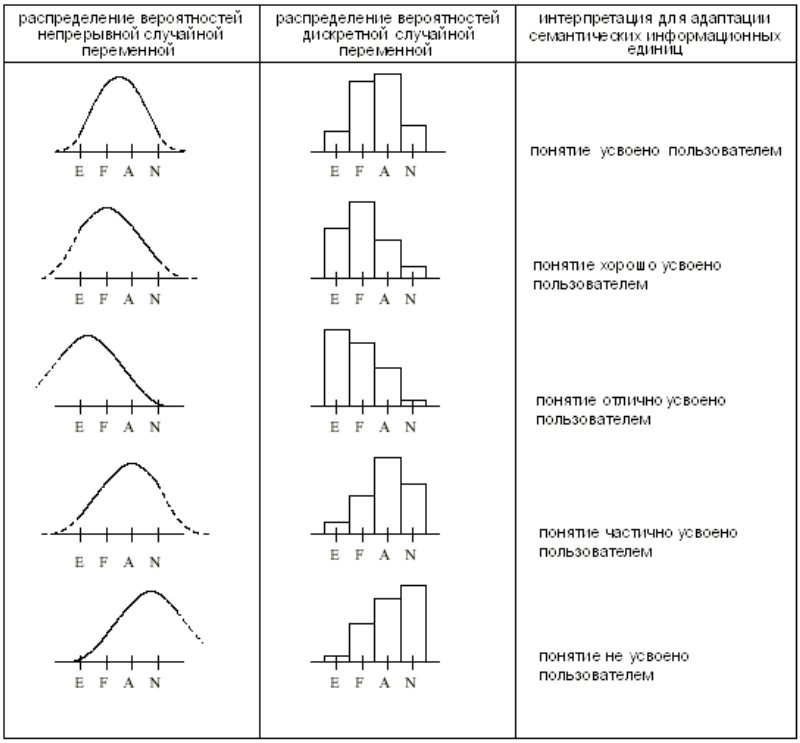


Рис. 3.5. Интерпретация распределений вероятности для сети Байеса

Интерпретация выводов сети Байеса. После того, как мы нашли механизм вывода для модели знаний, мы должны научиться интерпретировать результат вычисления: каким образом мы будем классифицировать знания пользователя как «уровень эксперта» или «уровень начинающего»? Сравнивая с непрерывной переменной, мы ищем максимум распределения (см. Рис. 3.5). К примеру, единица знаний K будет «хорошо усвоена» пользователем, если

$$P(K = \mathbf{E}) + P(K = \mathbf{A}) \geq P(K = \mathbf{F}) + P(K = \mathbf{N})$$

и «отлично усвоена» пользователем, если

$$P(K = \mathbf{E}) \geq P(K = \mathbf{F}) + P(K = \mathbf{A}) + P(K = \mathbf{N}).$$

3.6. Модель тестирования

Тесты — это вопросы студенту, ответы на которые оцениваются системой без участия преподавателя.

По виду различаются три типа тестовых вопросов: выборные, мультивыборные и наборные. *Выборный* тест предполагает выбор одного варианта ответа из предлагаемого списка (Рис. 3.6), а *мультивыборный* — нескольких вариантов (Рис. 3.7). *Наборный* тест требует, чтобы студент ввел правильный ответ в специальное текстовое поле (Рис. 3.8). По существу, каждый мультивыборный тест предлагает студенту список вариантов ответов, некоторое (возможно пустое) множество из которых является правильными.

Мы используем компьютерное тестирование со случайным позиционированием ответов для выборных и мультивыборных тестов, чтобы ответы по номерам позиций могли привести к ошибкам студентов, заучивающих позиционные номера правильных ответов. Таким образом, вместо запоминания вопросов и номеров строк вариантов правильных ответов студенты вынуждены в вопросах и ответах на вопросы фокусироваться на содержательной стороне дела. Поэтому подход к тестированию, используемый нашей системой, включает случайную генерацию вопросов в заданной предметной области и случайное расположение вариантов ответов.

Предполагается, что в процессе тестирования студент, как правило, получает не один, а целую серию тестов, ответы на которые оцениваются суммарно. Такой подход еще более усложняет работу тех студентов, которые пытаются пройти тестирование путем заучивания позиционных номеров правильных ответов.

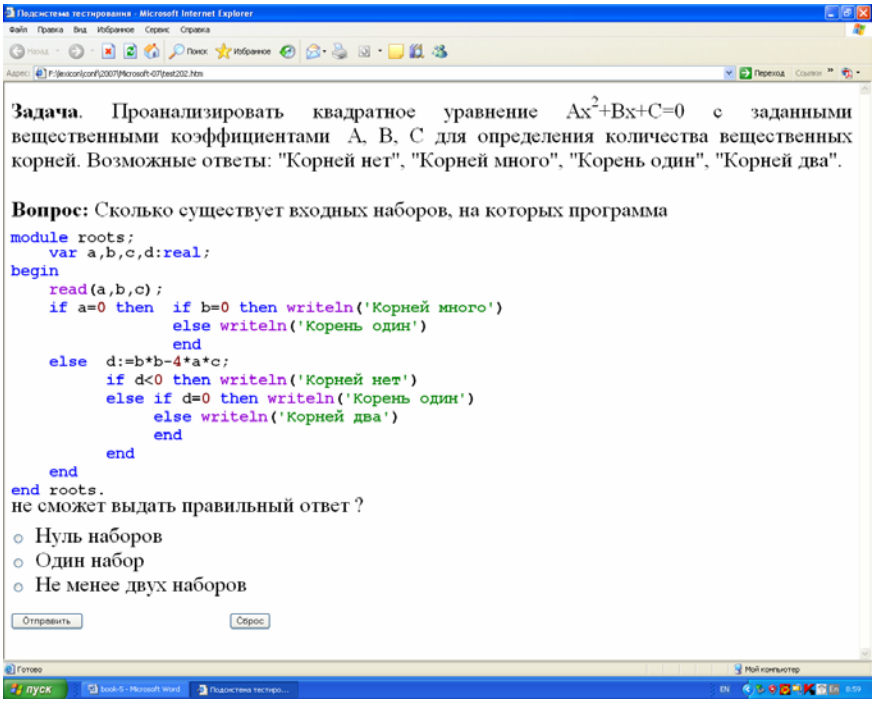


Рис. 3.6. Выборный тест

Каждый тест измеряет вербальные или аналитические умения, относящиеся к конкретной области изучаемого курса. Различные временные ограничения ассоциируются с каждым вопросом. Мы различаем три вида тестов: вербальные, качественные и аналитические.

Вербальный тест определяет некоторую конкретную концепцию определения и имеет временное ограничение в 60 секунд. Вербальный тест проверяет способность анализировать и оценивать написанный материал и синтезировать информацию, полученную из него, для анализа отношений между отдельными частями предложений и для распознавания отношений между словами и понятиями.

В случае *качественных* тестов, в которых должны быть объяснены более сложные понятия, обычно ответ ожидается между 120 и 240 секундами. Качественный тест проверяет базисные умения и понимание элементарных

понятий, а также способность студента качественно мыслить и решать задачи в качественной постановке или объяснять более сложные понятия.

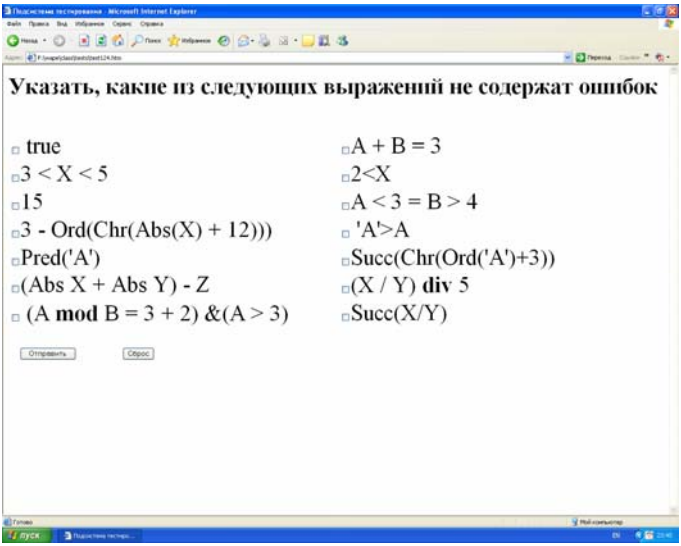


Рис. 3.7. Мультивыборный тест

Для *аналитического* теста, при котором должно быть объяснено некоторое трудное понятие, ответ обычно ожидается внутри временного диапазона в 360 — 480 секунд. Аналитический тест проверяет способность студента понимать структурированные множества отношений, выводить новую информацию из множеств отношений, анализировать и оценивать аргументы, идентифицировать центральные вопросы и гипотезы, делать правильные выводы и опознавать хорошо обоснованные объяснения. Тесты аналитического вида обычно измеряют умение рассуждать, связанное с несколькими разделами изучаемого курса.

С каждой единицей знания курса связываются два пространства тестов, первое из которых используется для тестирования студентов, претендующих на знание данной единицы знаний на уровне «начинающего», а второе — на уровне «продвинутого» студента.

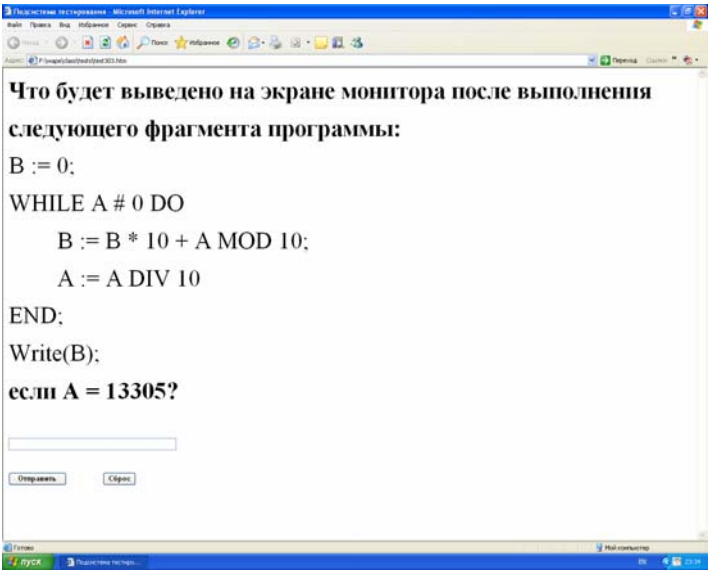


Рис. 3.8. Наборный тест

Пространство тестов — это ациклический ориентированный граф $G = (X, V)$, вершинами которого являются тесты, а дуги отражают последовательность их прохождения: $(p, q) \in V$ тогда и только тогда, когда после выполнения студентом теста p ему может быть предложен тест q.

В пространстве тестов выделяются множества входных и выходных тестов. Тест p является входным, если он не имеет предшественников в G, и является выходным, если p не имеет преемников в G (Рис. 3.9).

С каждым пространством тестов лектор связывает две константы границ знаний C_1 и C_2, где C_1 — минимальный уровень знаний для начинающего студента в данном пространстве тестов, а C_2 — минимальный уровень знаний для продолжающего студента для данного пространства тестов. По умолчанию предполагается, что пространство тестов для начинающих студентов состоит из вербальных и качественных тестов и имеет $C_2 > 1$, а пространство для продвинутых студентов состоит из аналитических тестов, и в нем $C_2 = 1$.

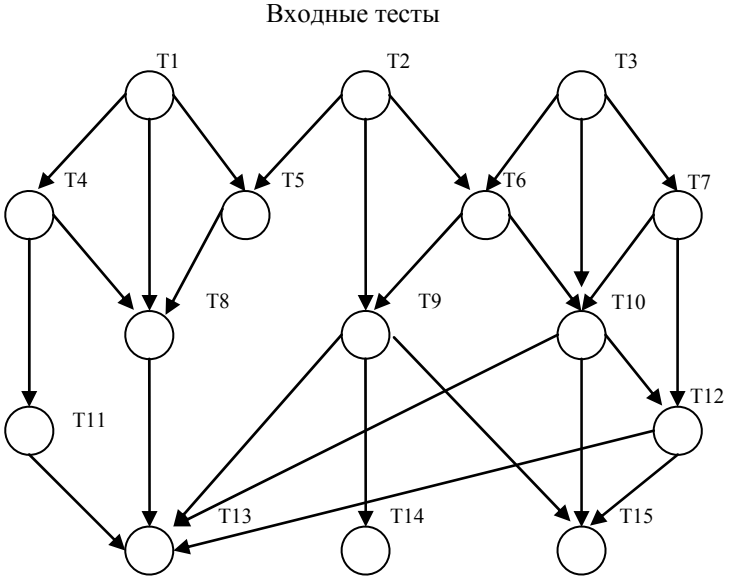


Рис. 3.9. Пространство из 15 тестов

Путь тестирования в пространстве тестов — это путь по G от некоторого входного теста до некоторого выходного. В процессе тестирования студенту предлагается серия тестов $T = \{p_1, \ldots, p_n\}$, образующих путь тестирования $(p_1, \ldots, p_n)$, генерируемый случайным образом. Например, последовательность (T3, T6, T9, T13) образует один из возможных путей тестирования для пространства тестов (Рис. 3.9).

Пусть студент выполнил серию тестов T и

$\gamma_x = \left(\sum_{t \in T} \alpha(t)\right) / \left(\sum_{t \in T} \beta(t)\right)$, где

- $\alpha(t)$ — число правильных ответов данного студента в серии тестов T,
- $\beta(t)$ — максимальное число правильных ответов для серии тестов T, которые мог бы дать студент. Заметим, что в случае выборного

или наборного ответа студент может дать только один правильный ответ, а в мультивыборном тесте максимальное число правильных ответов совпадает с длиной списка вариантов ответов.

Студент x демонстрирует на серии тестов T знания

- новичка, если $\gamma_x < C_1$,
- начинающего студента, если $C_1 \le \gamma_x < C_2$, и
- продвинутого студента, если $C_2 \le \gamma_x$.

3.7. Проекты системы CLASS

Системы CLASS и PRACTICE являются основными подсистемами системы WAPE.

Подсистема CLASS является виртуальной семинарской аудиторией, которая предназначена для получения опыта программирования на некотором языке высокого уровня. Это — среда проблемного обучения, в которой студенты группы под руководством инструктора обучаются конструировать корректные и эффективные программы для решения достаточно простых задач.

Любой курс, поддерживаемый системой CLASS, включает набор проектов, предназначенных для выполнения студентами. Каждый проект P — это упорядоченное множество однотипных задач $\{P_1, P_2,\ldots,P_n\}$, где $n \ge 0$ — количество вариантов проекта P.

В системе CLASS имеются проекты двух видов: упражнения и задания, различающиеся по виду предполагаемого решения.

Каждое *упражнение* P — это набор вопросов $\{P_1, P_2,\ldots,P_n\}$, предполагаемые ответы на которые не рассматриваются системой как исполняемые программы.

В отличие от упражнений любое *задание* P — это задача на программирование; так что решить вариант i задания P, т.е. P_i, — это значит написать исполняемую программу.

По уровню сложности решаемых задач различаются проекты двух типов: стандартной и повышенной сложности.

Проекты стандартной сложности не имеют пометок и образованы из тех задач, которые проверяют понимание студентом изложенного в учебнике теоретического материала. В частности, к проектам стандартной сложности относятся те упражнения и задания, которые направлены на обучение студентов использованию новых языковых конструкций совместно с уже усвоенными, а также на отработку описанных в учебнике схем решения задач.

Другой тип проектов выделен специальной пометкой «проект повышенной сложности» и содержит задачи, которые добавляют новую или требующую размышлений информацию к материалу, изложенному в учебнике. Такие проекты заставляют студентов обдумывать некоторую важную концепцию, относящуюся к теоретическому материалу учебника, или находить ответ на вопрос, который может возникать у студента во время чтения учебника.

Каждый проект стандартной сложности требует, чтобы у студента, допускаемого к его выполнению, уровень знания для всех тех единиц, которые образуют индекс данного проекта, был бы не ниже уровня продвинутого студента, а любой проект повышенной сложности может начать решать только тот студент, который является экспертом во всех единицах знаний, входящих в индекс данного проекта.

В каждом проекте можно выделить некоторый вариант в качестве примера. Решение такого варианта, который получает номер 0, оформляется в виде гипертекстового документа, содержащего формулировку задачи, ее решение и обоснование решения. В частности, в случае задания он содержит описание задания, текст программы и комментарии к программе, поясняющие и обосновывающие данное решение.

Для поддержки данной возможности используются виртуальные и пользовательские номера вариантов проектов.

Виртуальные номера используются при описании проекта лектором на языке описания проектов, а пользовательские — это номера, в терминах которых варианты данного проекта используются студентами в курсе, поддерживаемом системой CLASS. В частности, вариант, имеющий в качестве пользовательского номера нуль, является примером данного проекта.

Лектор, описывая проект в терминах виртуальных номеров, одновременно задает функцию преобразования виртуальных номеров в пользовательские. Указанная функция используется подсистемой генерации проектов каждый раз, когда происходит обращение к проекту за формулировкой некоторого его варианта или за его тестами.

3.8. Задания системы CLASS

Каждое задание системы CLASS предполагает составление программы, читающей входные данные из файла *input.txt* и записывающие результаты своего исполнения (выходные данные) в файл *output.txt.*

С каждым заданием α связывается множество эталонных решений $S(\alpha)$, множество тестов задания $T(\alpha)$, множество правил вывода знаний студента

$R(\alpha)$ и два вектора знаний: $P(\alpha)$ — вектор предварительных знаний и $F(\alpha)$ — вектор итоговых знаний.

Эталонные решения $S(\alpha)$ — это прокомментированные программы решения задания α. В комментариях описываются особенности данного решения задания α и приводится мотивированная его оценка. Эталонные решения используются преподавателями (лекторами и инструкторами) и не доступны студентам.

Тесты задания $T(\alpha)$ используются системой для проверки правильности понимания студентом условия задания и автоматической проверки правильности составленной студентом программы. Каждый тест из множества $T(\alpha)$ — это пара, состоящая из входных и соответствующих выходных данных.

Множество тестов $T(\alpha)$ задания α распадается на два непересекающихся подмножества $T_1(\alpha)$ и $T_2(\alpha)$, т.е. $T(\alpha) = T_1(\alpha) \cup T_2(\alpha)$ и $T_1(\alpha) \cap T_2(\alpha) = \varnothing$, называемых тестами *правильности понимания задания* и тестами *правильности программы* (решения задания).

Перед тем, как начать решать задание, студент должен подтвердить, что он правильно понимает условие задания. Для этого он должен для каждого теста $t \in T_1(\alpha)$ правильно ввести выходные данные теста t по заданным входным. Тесты из $t \in T_1(\alpha)$ могут использоваться студентом также на начальном этапе отладки программы решения задания α.

Перед тем, как студенту будет разрешено отдать программу (решение задания α) на проверку инструктору, он должен продемонстрировать системе ее правильность на всех тестах из $T_2(\alpha)$. Для этого его программа должна на входных данных каждого теста $t \in T_2(\alpha)$ выдавать выходные данные, которые соответствуют ожидаемым.

Каждому тесту $t \in T(\alpha)$ могут быть сопоставлены два текста: диагностическое сообщение и общедоступный комментарий. *Диагностическое сообщение* — это тот текст, который выдается студенту тогда, когда программа студента является неправильной относительно теста t. В тексте комментария указываются предполагаемые ограничения на рабочие характеристики программы для данного теста.

Каждое *правило вывода* знаний студента из множества $R(\alpha)$ имеет вид: $B \to N$, где

B — логическое выражение, содержащее литералы вида x или $\neg x$,

$x \in T_2(\alpha)$, а

N — вектор знаний.

Правило $B \to N$ работает в два этапа.

На первом этапе в B вместо каждого вхождения теста $x \in T_2(\alpha)$ подставляется либо значение *true,* если на входных данных теста x программа выдает предполагаемые результаты, либо значение *false* в противном случае.

На втором этапе вычисляется выражение B, и, если оно принимает истинное значение, то происходит перевычисление вектора знаний $K(x)$ студента x, решающего данное задание, по следующему правилу $K(x) := min(N, K(x))$.

Вектора предварительных и итоговых знаний $P(\alpha)$ и $F(\alpha)$, приписанные заданию α, имеют следующий смысл. Предполагается, что студент x может приступить к выполнению проекта α только в том случае, если $K(x) \geq P(\alpha)$. Успешное выполнение проекта α приводит к перевычислению вектора знаний $K(x)$ студента x, решающего данное задание, по следующему правилу $K(x) := max(F(\alpha), K(x))$.

3.9. Система PRACTICE

Система PRACTICE является виртуальной лабораторией, предназначенной для прохождения студентами компьютерного практикума по курсу. Основная цель, которая ставится перед студентом при выполнении индивидуальных заданий (интегральных проектов), составляющих компьютерный практикум, — это практическое освоение всех этапов разработки надежной и наглядной интерактивной (диалоговой) программы для компьютерного решения некоторой нетривиальной задачи, требующей разработки алгоритма, обработки сложных структур данных и создания дружественного интерфейса.

С каждым интегральным заданием α связывается множество эталонных решений $S(\alpha)$ и два вектора знаний: $P(\alpha)$ — вектор предварительных знаний и $F(\alpha)$ — вектор итоговых знаний. Они имеют тот же смысл, что и у заданий системы CLASS, с тем отличием, что *эталонные решения* $S(\alpha)$ — это прокомментированные решения задания α, которые имеют вид гипертекстовых отчетов (см. ниже).

Тематика индивидуальных заданий для компьютерного практикума определяется, в первую очередь, всеми видами работ, которые должен освоить студент, чтобы научиться создавать качественные (эффективные, наглядные и надежные) нетривиальные программы. В большинстве случаев задача, решаемая во время выполнения индивидуального задания, — это задача невычислительного характера, имеющая краткую и точную (содержательную) формулировку и допускающая большое разнообразие решений.

При выполнении проектов практикума студенты должны интегрировать все, что они изучили ранее (при работе в системе CLASS), а также получить навыки, являющиеся базовыми для разработки программного обеспечения на всех уровня и для программирования как дисциплины. Это включает использование в той или иной мере формальных методов анализа задач и конструирования программ с упором на создание эффективных и надежных программ, которые удовлетворяют заданным спецификациям и поддерживают дружественный интерфейс.

Работая в виртуальной лаборатории, студент изучает методический материал, содержащийся в задачнике курса, тестирует свои знания теоретического материала, прочитанного им, или решает интегральные проекты.

В процессе выполнения проекта в системе PRACTICE студент составляет гипертекстовый отчет, который включает:

1) формулировку задачи;

2) описание программы для пользователя, ее внешнюю спецификацию, т.е. описание способа задания входных данных, вида результатов программы при заданных входных данных и сценария диалога в процессе исполнения программы;

3) словесное описание алгоритма и обоснование его правильности и эффективности;

4) текст программы;

5) описание тестового набора и его обоснование.

3.10. Аннотация проектов

Для вычисления релевантности доступного по ссылке проекта текущему уровню знаний студента используется простая метафора «светофора» для аннотаций. Ссылки на проекты, которые должен выполнить в курсе студент, помечены одним из трех цветовых кружков: готов-для-решения (зеленый кружок рядом с текстом ссылки), не-готов-для-решения (красный кружок) или уже-решен (серый кружок), что позволяет студенту выбрать подходящие проекты.

Студенту часто требуется информация по определенным темам, но не хватает предварительных знаний для понимания этой информации. К примеру, студент хочет работать над проектом построения ветвящихся программ, но не понимает таких тем, как логические выражения; в таком случае ему не стоит начинать чтение сразу со страниц, содержащих описание условных операторов. Для поддержки студента система сравнивает его текущий уровень знаний с необходимым для понимания рассматриваемой

темы. Если у студента не хватает некоторых предварительных знаний, система может сгенерировать след — последовательность информационных элементов, который направляет его обучение в рамках данной темы.

Генерация такого следа реализована как алгоритм обхода в глубину, который проверяет оценку системой знаний студента о тех единицах знаний, которые предварительно необходимы для его текущей цели. Алгоритм проверяет, все ли предварительные знания достаточно усвоены студентом. Если нет, алгоритм находит единицы знаний, нуждающиеся в изучении. После этого генерируется последовательность подцелей и последовательность информационных элементов, которые последовательно направляют работу студента через необходимые темы вплоть до выбранной им.

Если студент хочет получить более конкретные указания во время работы с системой, он может запросить у системы следующий разумный шаг обучения. Этот запрос на прямое руководство выполняется путем определения подходящей цели обучения, зависящей от текущего состояния знаний студента и состояния его проектов. Цель определяется как набор единиц знаний. Чтобы определить следующую подходящую цель обучения, вычисляется последовательный след, покрывающий все нерешенные проекты студента. Для каждого элемента этого следа проверяется оценка системой уровня знаний студента. Если студенту не хватает знания какой-либо единицы знаний, тогда она предлагается в качестве следующей подходящей цели.

3.11. Режимы работы студента

Инструктор может управлять тем, как знания студента влияют на возможность его работы с системой.

В *жестко контролируемом* режиме студент x не может выполнять ни один из тех проектов, для которых он не имеет достаточно знаний. Другими словами, при этом режиме разрешается студенту x приступить к выполнению некоторого проекта α только в том случае, когда $K(x) \geq P(\alpha)$.

В *слабо контролируемом* режиме студент x может выполнять не только те проекты α, для которых у него хватает знаний (т.е. $K(x) \geq P(\alpha)$), но и любой такой проект α, что для любого i либо $K(x)(i) \geq P(\alpha)(i)$, либо $K(x)(i)=F$ и $P(\alpha)(i)= E$.

В *свободном режиме* студент может выполнять любой проект.

При этом вне зависимости от того режима, который установлен студенту x инструктором, студент всегда получает предупреждение о недостаточности своих знаний, если он пытается начать выполнять такой проект α,

для которого у него не хватает компетентности в некоторой единице знаний i (т.е. $K(x)(i) < P(\alpha)(i)$). Одновременно с предупреждением студенту демонстрируются те единицы знаний (вместе с их уровнями), знание которых он должен повысить до выполнения данного проекта.

3.12. Обновление модели знаний студента

Многие адаптивные системы фиксируют факт чтения пользователем определенной информации и на этой основе обновляют оценку его знаний. Некоторые из них учитывают время чтения или последовательность прочитанных страниц для углубления этой оценки. Хотя это оправданный подход, его недостатком является трудность измерения знания, приобретенного пользователем во время чтения Web-страницы. Вместо этого мы используем для обновления модели знаний только тесты и проекты. Это мотивировано проблемным подходом к обучению в тех курсах, на поддержку которых ориентирована система WAPE.

Предполагается, что после чтения того или иного теоретического материала система организует обратную связь. Студент последовательно указывает те темы, которые были целью изучения данного материала. Для каждой из этих тем (единиц знаний) студент переоценивает свои изменившиеся знания, выбирая одну из двух категорий: «тема была несложной — я овладел материалом без труда», «тема была непростой — у меня были некоторые проблемы в понимании». Эти категории соответствуют знаниям продвинутого студента и начинающего. После этого система генерирует индивидуальный набор тестов, успешное прохождение которых позволяет студенту изменить желаемым образом оценку своих знаний.

Обновление модели знаний автоматически происходит во время работы студента над некоторым проектом каждый раз, когда решение студента не проходит некоторый тест, а также тогда, когда инструктор завершает оценку текущего (посылаемого инструктору на проверку) варианта решения проекта студентом.

Студент имеет право в любой момент понизить оценку уровня своих знаний любой единицы знаний, а также попытаться повысить её. В последнем случае в зависимости от его претензий он получает индивидуальный набор тестов, сгенерированный системой, успешное прохождение которых позволяет студенту изменить желаемым образом оценку своих знаний.

3.13. Развитие курса

В процессе функционирования предполагается развитие каждого курса, поддерживаемого системой, по следующим направлениям:

1) включение новых учебников в курс,

2) создание или улучшение примеров решения проектов,

3) пополнение заданий новыми тестами и эталонными решениями.

Указанные усовершенствования осуществляются лектором и не требуют от него каких-либо специальных программистских знаний. При включении некоторого нового гиперучебника в курс нет необходимости его преобразования. Единственное, что требуется — это построение информационных индексов элементарных информационных элементов, составляющих данный учебник. Эти индексы можно строить как автоматически (путем поиска вхождений ключевых фраз), так и вручную (путем просмотра учебника лектором). В основе других направлений развития курса лежат наработки студентов, которые при включении их преподавателем в курс могут подвергнуться редактированию и комментированию.

Для описания новых проектов можно использовать следующий специальный *язык задания проектов* (ЯЗП), позволяющий задавать одновременно все варианты проекта компактным образом.

Ниже для описания синтаксиса языка ЯЗП используется Расширенный Бекуса-Наура Формализм (Extended Backus-Naur Formalism, EBNF), который характеризуется следующими свойствами:

- альтернативы разделяются символом |;
- скобки [и] обозначают факультативность выражения в скобках;
- скобки { и } обозначают повторение (возможно 0 раз);
- скобки (и) используются для формирования групп элементов;
- нетерминальные символы начинаются с прописной буквы (например, Statement);
- терминальные символы либо начинаются со строчной буквы (например, letter), либо записывается целиком прописными буквами (например, BEGIN), или представляются в виде строк (например, ":=");
- комментарии начинаются с символа // и продолжаются до конца строки.

Правила описания проекта на языке ЯЗП имеют следующий вид, где Символ — это произвольный символ, отличный от знака # :

Проект={Условие Образец}.

Образец={{Символ}{#Выражение}{Символ}}.

Условие= ИстинноеУсловие | ВычисляемоеУсловие.
ВычисляемоеУсловие = #<Выражение, Интервал> |
#<Выражение, Число>.
ИстинноеУсловие = ##.
Выражение= Номер | Номер * Число | Номер DIV Число.
Интервал=ОткрывСкобка Число, Число ЗакрСкобка.
ОткрывСкобка = (| [.
ЗакрСкобка =) |].
Число = {Цифра}.
Номер = # Число.

Каждый Проект на языке ЯЗП представляет собой фактически последовательность строк вида Условие Образец, по которым строятся все конкретные варианты проекта по следующим правилам. Вначале *N* подставляется в каждое Выражение вместо Номера, преобразуя их в константные. Затем полученные константные выражения вычисляются, и вычисленные значения заменяют вхождения выражений. Данная подстановка преобразует образцы в строки факультативных и обязательных частей текста формулировки соответствующего варианта. Обязательная часть образуется из образца, которому предшествует ИстинноеУсловие. Условие, соответствующего образца, при котором данная факультативная строка должна включаться в формулировку конкретного варианта проекта, — это принадлежность вычисленного значения выражения интервалу, заданному в вычисляемом условии, или его равенство заданному значению.

3.14. Реализация системы WAPE

Экспериментальная реализация первой версии системы WAPE выполнялась в 2002–2005 гг. [25, 26]. В разработке ее отдельных подсистем участвовали студенты [4, 5].

В отличие от описанного в данной главе проекта первая версия системы имела архитектуру, приближенную к модельной архитектуре адаптивных систем обучения общего вида, описанной в гл. 2 и состоящей из моделей предметной области, пользователя и адаптации. В ней использовалась модель предметной области в виде концептов и межконцептных отношений, оверлейная модель пользователя, в которой знания пользователя связывались с посещением соответствующих страниц гиперучебников и выполнением связанных с ними тестов, а также модель адаптации в виде системы конкретных и обобщенных правил адаптации, предполагающих специаль-

ный вид представления гиперучебников и их адаптацию в процессе работы системы.

Для включения курса в систему гиперучебники, поддерживающие курс, должны быть представлены на специально разработанном языке, базирующемся на ссылочной модели AHAM [91], которая является расширением хорошо известной ссылочной модели Dexter для гипермедиа [103]. Курс, который встраивался в систему, — это вводный курс программирования на базе языка Паскаль [16, 38].

Экспериментальная реализация показала, что такая архитектура делает задачу включения нового курса в систему весьма трудоемкой задачей и является малополезной при его использовании с точки зрения настройки процесса обучения на нужды обучаемого и оценки его знаний.

Другое важное отличие первой версии системы от рассмотренной выше состояло в том, что она позволяла транслировать и исполнять программы в рамках системы.

Экспериментальные исследования показали, что такая возможность является малооправданной в силу многозадачности используемой студентами системы Windows и приводит к сложным проблемам, связанным с безопасностью системы.

Выводы по главе 3

В главе описывается архитектура расширяемой адаптивной среды дистанционного обучения, которая поддерживает активное индивидуальное обучение программированию в рамках проблемного подхода и соединяет возможности адаптивных гипермедиа-систем и интеллектуальных обучающих систем. Среда нацелена на поддержку обучения конструированию алгоритмов и разработки эффективных и надежных программ, в процессе которого обучаемый, решая поставленные ему индивидуальные задачи, действует вполне самостоятельно, но постоянно имеет возможность получения квалифицированной помощи, корректирующей и направляющей его усилия, начиная с этапа понимания условия задачи и кончая этапом оценки правильности решения.

Мы описали модель обучаемого и методы мониторинга его знаний, которые поддерживают адаптивность в рамках проблемного подхода к обучению и позволяют оценивать знания обучаемого в условиях неполной информации с использованием сетей Байеса. Многие адаптивные обучающие системы в модели студента отслеживают его движение по гиперкниге. Хотя это оправданный подход, его недостатком является трудность измерения

знания, приобретенного студентом во время чтения Web-страницы. Вместо этого мы используем для обновления модели знаний только успехи и неуспехи студента, проявленные им при решении задач: тестов, заданий и упражнений. Другое важное отличие рассмотренного подхода от традиционного состоит в том, что мы не пытаемся оценивать успех студента в изучении курса на основании уровня знаний в его модели. Поэтому в нашей системе достижение определенного уровня знаний студентом не позволяет ему завершить изучение курса с определенной оценкой, а лишь дает студенту возможность приступить к решению той или иной задачи из его индивидуального набора. Все это мотивировано проблемным подходом к обучению, поддерживаемым системой WAPE.

Тесты — это вопросы студенту, ответы на которые оцениваются системой без участия преподавателя. Представленные в данной главе модель и методы тестирования знаний обучаемого охватывают известные подходы к тестированию знаний студента и позволяют генерировать сценарий тестирования случайным образом. В основе модели лежит разработанная классификация тестов по форме (выборные, мультивыборные, наборные) и по глубине проверяемых знаний и временным ограничениям на выполнение тестов (вербальные, качественные и аналитические), а также объединение тестов в так называемые пространства тестов по содержанию проверяемого знания.

ГЛАВА 4.
ВВОДНЫЙ КУРС ПРОГРАММИРОВАНИЯ НА БАЗЕ ЯЗЫКА ZONNON

При обучении программированию наиболее важным представляется начальный этап, на котором обучаемый должен овладеть навыками точного формулирования алгоритмов на языках высокого уровня. Это невозможно сделать, прочитав несколько руководств или прослушав курс лекций по программированию. Необходима практика конструирования алгоритмов, и здесь невозможно обойтись без языка программирования, пригодного для целей начального обучения, и подходящего набора примеров и задач.

В главе кратко представлен вводный курс программирования на базе языка Zonnon, описаны две книги: «Введение в программирование» и «Практикум по программированию» [20, 23], поддерживающие этот курс.

Zonnon — это новый универсальный язык программирования в семействе языков Паскаль, Модула-2 и Оберон, работа над которым ведется в Цюрихском институте информатики [27, 101, 102]. Он сохраняет стремление к простоте, ясному синтаксису и независимости концепций, а также уделяет внимание параллельности и легкости композиции и выражения. Унификация абстракций является стержнем проектирования языка Zonnon, и она отражается в его концептуальной модели, основанной на модулях, объектах, определениях и реализациях. Язык Zonnon содержит такие новые черты, как активность в объектах, основанный на межобъектном взаимодействии диалог, перегрузка операций и обработка исключительных ситуаций. Язык Zonnon специально разрабатывается как платформно-независимый язык. Первая реализация языка Zonnon выполнена для платформы .NET [156]. Кроме того, предполагается интеграция компилятора в систему программирования Visual Studio .NET (в сотрудничестве с компанией Microsoft) [10].

Глава начинается с изложения основных особенностей языка Zonnon (разд. 4.1). Разд. 4.2 содержит описание цели и основных принципов курса. Разд. 4.3 является кратким описанием книги «Введение в программирование», являющейся учебником по курсу. Цели и содержание компьютерного практикума составляют разд. 4.4. Разд. 4.5 содержит описание книги «Практикум по программированию», предназначенной для поддержки практикума. Вопросы использования курса в рамках системы WAPE рассматриваются в разд. 4.6. Главу завершают выводы по главе.

Полное описание языка Zonnon в его текущей версии приведено в приложении. Компилятор и среду программирования для данной версии языка можно скачать с сайта разработчиков языка Zonnon [156]. Там же всегда можно найти последние новости о языке, его реализации и применении, а также обратиться за консультацией к разработчикам.

4.1. Язык программирования Zonnon

Сам проект по разработке языка Zonnon возник как продолжение работы его авторов над языком Оберон (Oberon) в проекте, который был начат Microsoft Research в 1999 г. с целью реализации значительного числа нестандартных языков программирования для платформы .NET [52, 55]. Язык Оберон [151, 152] является хорошо известным преемником языков Паскаль (Pascal) и Модула-2 (Modula-2). Мотивация авторов на продолжение работы по развитию языка Оберон связана со следующими двумя целями [101]:

- экспериментально исследовать потенциальные возможности платформы .NET в комбинации с новой технологией CCI интеграции компиляторов для проектирования языков;
- реализовать для платформы .NET язык Zonnon, являющийся эволюцией языка Oberon для .NET.

Поскольку язык Zonnon задуман как дальнейшая эволюция языка Оберон, авторы стремятся сохранить такие важные черты языка Оберон и его преемников, как компактность языка, ясность, недвусмысленность и ортогональность его основных понятий. Вместе с тем, чтобы создать современную альтернативу языку Оберон, авторы внесли в язык ряд изменений, основными из которых являются:

- более развитая модульная структура языка;
- продвинутая и одновременно простая и ясная объектная модель;
- концепция активных объектов.

«Общеалгоритмическая» часть языка Zonnon практически полностью повторяет соответствующие части языка Оберон, поэтому данный язык можно рассматривать как естественную замену Оберону там, где последний традиционно используется для обучения программированию.

Язык Zonnon возник как естественный результат исследований, проводившихся в течение последних нескольких лет в Федеральном Технологическом Институте (ETH) в Цюрихе. Непосредственными предшественниками языка следует считать Active Oberon, реализованный как базовый язык ядра операционной системы BlueBottle, а также реализацию языка Oberon для платформы .NET (Oberon.NET). Язык Zonnon, имея ряд общих

черт с указанными языками, вместе с тем существенно отличается от них по ряду принципиальных моментов. Первая реализация Zonnon выполнена для платформы .NET. Кроме того, предполагается интеграция компилятора в систему программирования Visual Studio .NET (в сотрудничестве с компанией Microsoft).

Хотя по размеру Zonnon уступает таким языкам, как C#, Java и Ada (см. приложение), он является универсальным языком программирования, пригодным для широкой области приложений. Типично она включает компонентно-ориентированную композицию, параллельные системы, алгоритмы и структуры данных, объектно-ориентированное и структурное программирование, графику, математическое программирование и низкоуровневое системное программирование. Zonnon предоставляет богатую объектную модель с инкапсулированным поведением и синтаксически управляемыми диалогами, которые инкапсулируют состояние. Он может использоваться для написания программ в традиционном и объектно-ориентированном стилях, а также весьма подходит для целей обучения программированию от его базовых принципов до продвинутых концепций.

Унификация абстракций является стержнем проектирования языка Zonnon. Она отражается в его четырех столпах:

- *модуль (module)* — текстовый контейнер, а также объект композиции программ;
- *объект(object)* — типовой образец для определяемых объектов;
- *определение (definition)* — концепция абстракции и композиции для определяемых интерфейсов;
- *реализация (implementation)* — контейнер для переиспользуемых фрагментов объектных реализаций.

Модуль (*module*) имеет двойственную природу: он объявляет синтаксический контейнер для логически связанных программных деклараций и одновременно определяет объект, чей жизненный цикл управляется системой. Таким образом, модуль предоставляет механизм для текстуального разбиения исходной программы, а также динамической загрузки на этапе выполнения некоторой части программы в виде созданного экземпляра объекта.

Любое число динамически созданных объектов может иметь свои жизненные циклы, управляемые системой, однако в любое заданное время только единственный экземпляр каждого модульного объекта может быть создан системой. Поскольку модуль формирует единицу инкапсуляции и сокрытия данных, он представляет собой идеал как контейнер для реализации абстрактных типов данных.

Объектом (*object*) является типовой образец, включающий в себя поля, методы и активности. Поля представляют состояние объекта, методы — его функциональность, а активности — его параллельное поведение. Он может выставлять свой интерфейс к системному окружению двумя способами. Во-первых, с помощью присущего ему интерфейса (*intrinsic interface*), т.е. множества всех элементов, которые программист выбрал сделать публичными, а не сохранять как приватные, и, во-вторых, большим числом определений (*definitions*), каждое из которых выставляет свой аспект (*facet*), представляющий собой некоторый взгляд на службы объекта со стороны клиентов.

Определение (*definition*) задает некоторый свой аспект (*facet*) объекта в терминах абстрактного интерфейса, включающего определения полей и сигнатуры методов. Определения могут образовывать не только иерархию, но и сеть связанных типов (*a network of related types)*.

Реализация (*implementation*) определяет агрегат реализаций полей и методов, предназначенных для переиспользования, при присоединении к программе с помощью одного или нескольких объектных образцов. Объект, реализующий определение, должен реализовывать все его поля и методы. Однако если объект импортирует реализацию определения с тем же именем, то это неявно предполагает, что она будет его (возможно частичной) реализацией.

Программный текст (*program text*) состоит из модулей, объектов, определений и реализаций. Внутренний интерфейс (*intrinsic interface*) программы представляет собой множество деклараций, сделанных публичными всеми ее частями. Программа периода исполнения (*runtime program*) состоит из одного или более модулей и любых объектов, которые создаются динамически. Система (*system*) предоставляет механизмы динамической программной загрузки и выгрузки модулей и динамическим управлением объектных ресурсов во время выполнения, когда происходит исполнение программы.

Эти конструкции используются для формирования всей структуры программы в виде программных *единиц: module, object*, *definition* и *implementation*. Каждая конструкция может существовать как отдельно скомпилированная единица (*separately compiled unit*) или может текстуально встраиваться в содержимое другой конструкции. Указанные единицы обеспечивают базис для композиции программ при «программировании в большом», а также для текстуального разбиения и раздельной компиляции во время разработки программы.

Объектная модель в языке Zonnon основывается на концепции, что «все есть объект». Она поддерживает три точки зрения на объекты: во-первых, как сущности с некоторым внутренним типом, использующие абстрактные операции способом, безопасным для типов, во-вторых, как провайдеры служб, доступные через определенные интерфейсы и, в-третьих, как автоматические агенты, взаимодействующие через формальные диалоги. *Активности (activities)* используются как для добавления поведения объектам, так и для реализации диалога. Они внедряют естественным образом параллелизм в язык.

Многие из концепций языка Zonnon получены им по наследству. Цель состояла в том, чтобы предложить выразительные и связующие черты, которые уже доказали свою ценность. Язык Zonnon также вводит некоторые новые черты, такие как перегрузка знаков операций для естественного представления математических и других выражений и обработка ситуаций для повышения надежности. Некоторые черты были им реанимированы (взяты из ранних членов семейства паскалеподобных языков), например, пары *definition*, *implementation* и типы перечисления языка Модулы-2 и, по прагматическим причинам, основная форма операторов вызова процедур *read* и *write* языка Паскаль.

Когда осуществляется выбор языка для построения современных программных систем, важным соображением является достижение взаимодействия между программами, написанными на разных языках. Язык Zonnon специально проектировался как платформно-независимый язык, поддерживающий такое взаимодействие.

Ниже некоторые из основных новых языковых понятий будут проиллюстрированы на небольшом наборе простых, но типичных примеров, приведенных авторами языка [101].

Определения и реализации можно пояснить на следующем примере. Патефон-автомат имеет две “грани” (“facets”): его можно воспринимать либо как хранилище записей, либо как проигрыватель. Соответственно, можно использовать следующие определения:

```
DEFINITION Store;
PROC Clear;
Add (s: Lib.Song);
END Store.
DEFINITION Player;
VAR cur: Lib.Song;
PROC Play (s: Lib.Song);
PROC Stop;
END Player.
```

Предположим, что, кроме того, имеется следующая реализация Store по умолчанию

```
IMPLEMENTATION Store;
VAR rep: Lib.Song;
PROC Clear;
BEGIN rep := NIL
END Clear;
PROC Add (s: Lib.Song);
BEGIN s.next := rep; rep := s
END Add;
BEGIN Clear
END Store.
```

Тогда можно использовать следующее определение объекта патефона-автомата:

```
OBJECT JukeBox IMPLEMENTS Player, Store;
IMPORT Store; (* aggregate *)
PROCEDURE Play (s: Lib.Song);
IMPLEMENTS Player.Play;
PROCEDURE Stop IMPLEMENTS Player.Stop;
..
END JukeBox.
```

Заметим, что реализация Store по умолчанию неявно агрегируется с пространством объектного состояния.

Активные объекты можно проиллюстрировать на следующем примере. В некотором простом террариуме поддерживается следующий вид поведения содержащихся в нем живых тварей. Если температура опускается ниже некоторого определенного минимума, представители всех видов погружаются в спячку, иначе они либо перемещаются случайным образом, либо, если голодны, требуют помощи.

```
OBJECT Creature;
VAR X, Y, temp, hunger, kill: INTEGER;
PROCEDURE NEW (x, y, t: INTEGER);
BEGIN X := x; Y := y; temp := t;
hunger := 0
END;
PROCEDURE SetTemp (dt: INTEGER);
```

```
BEGIN {EXCL} temp := temp + dt
END SetTemp;
BEGIN {ACTIVE}
LOOP
AWAIT temp >= minTemp;
WHILE hunger > minHunger DO
HuntStep(5, kill);
hunger := hunger – kill;
WHILE (kill > 0) & (hunger > 0) DO
HuntStep(7, kill);
hunger := hunger – kill
END;
RandStep(2)
END;
RandStep(4); hunger := hunger + 1
END
END Creature;
```

Заметим, что тело определения объекта последовательно описывает полную историю жизни таких живых тварей. Также заметим, что их поведение еще зависит существенным образом от среды, вызывающей метод `SetTemp`. В частности, любой экземпляр существа, находящегося в террариуме, может быть заблокирован оператором WAIT до тех пор, пока температура не поднимется выше минимума.

Модули — это объекты системного уровня, чей жизненный цикл управляется системой автоматически. В частности, модуль динамически загружается в момент первого вызова. Модули относятся к “статическим” объектам, которые статически могут импортировать другие модули. Хорошим примером системно-ориентированных модулей является менеджер ресурсов. Следующий набросок показывает менеджера окнами с инкапсулированной структурой данных, которая представляет текущую конфигурацию окна в дисплейном пространстве системы. Заметим, что менеджер окнами содержится в пространстве имен, называемом *System*, и что он базируется на другом модуле, называемом `DisplayManager`.

```
MODULE System.WindowManager;
IMPORT System.DisplayManager;
(* static import *)
OBJECT {VALUE} Pos;
VAR X, Y, W, H: INTEGER
END Pos;
```

```
DEFINITION Window;
VAR pos: Pos;
PROCEDURE Draw ();
END Window;
VAR {PRIVATE} W, H: INTEGER;
bot: OBJ { Window };
PROCEDURE Open(this:OBJECT{Window},p:Pos);
BEGIN ...
END Open;
PROCEDURE Change(this:OBJECT{Window},p:Pos);
BEGIN ...
END Change;
BEGIN (* module initialization *) bot:=NIL;
W := System.DisplayManager.Width();
(* delegation *)
H := System.DisplayManager.Height();
END WindowManager.
```

Структурно унифицированно можно представлять систему периода исполнения как ациклическую иерархию модулей. Обычно внизу и наверху иерархии расположены системные модули и модули приложений соответственно.

4.2. Цель и основные принципы курса

Курс предназначен для обучения основным методам построения корректных, эффективных и надежных программ на базе языка Zonnon и платформы Microsoft.NET. Он ориентирован на широкий круг лиц, обучающихся методам программирования, в первую очередь, на студентов НГУ, других вузов и средних учебных заведений, а также школьников, желающих углубить свои знания по программированию. Курс предназначен главным образом для тех учебных заведений, в которых в настоящее время используется язык Паскаль в качестве языка начального обучения программированию и есть желание перейти к более современному курсу программирования, охватывающему концепции языков программирования нового поколения, таких как Java и C#, но осуществить этот переход плавно, без резкого изменения сложившегося стиля преподавания программирования.

Курс опирается на опыт преподавания основного курса по программированию для студентов механико-математического факультета (ММФ) Новосибирского государственного университета (НГУ) [16]. Это семестровый курс, который предполагает еженедельно шесть часов аудиторных занятий:

- два часа лекционных занятий,
- два часа семинарских занятий у доски и
- два часа практических занятий за компьютером.

В основу курса положен принцип концентрического изложения материала. Предполагается, что с первых занятий студенты начинают упражняться в составлении программ, которые могут реально выполняться на доступных ЭВМ, и постепенно овладевают навыками разработки на языке Zonnon линейных, ветвящихся и итеративных алгоритмов, алгоритмов с массивами и процедурами. Также постепенно, одновременно с расширением класса решаемых задач, студенты углубляют свои знания о языке Zonnon.

К концу вводной части курса студенты овладевают основными конструкциями языка, образующими то естественно выделяемое его ядро для «начального обучения», которое условно можно назвать языком мини-Zonnon. В этом ядре нет ряда весьма важных возможностей языка Zonnon, таких как, например, поддержка объектно-ориентированного программирования, сужены средства языка Zonnon по структурированию данных и действий — например, нет множеств и бесконечных циклов. Поэтому, хотя мини-Zonnon и содержит все необходимые сведения о языке Zonnon для построения программ начального обучения, но он является лишь базой для изучения программирования на языке Zonnon. Отметим, что в мини-Zonnon идентификаторы могут содержать наряду с латинскими буквами и буквы русского алфавита, хотя это и запрещено в языке Zonnon.

Следует отметить, что использование в качестве основы вводной части курса подмножества «живого» языка программирования на ММФ НГУ является традиционным. Первым использовалось подмножество языка Алгол-60, получившее название языка Минал [18]. Затем, когда в НГУ появились терминальные классы с диалоговым языком, являющимся вариантом языка Фортран, Минал был заменен языком мини-Фортран [51]. В начале 80-х гг. прошлого столетия в НГУ появилась одна из первых реализаций языка Паскаль, тогда еще на машинах серии ЕС [47], и с тех пор вплоть до настоящего времени место первого учебного языка программирования для студентов ММФ НГУ бессменно занимает язык мини-Паскаль [16].

Другими основными методическими и технологическими принципами, на которых базируется разработанный курс, являются следующие.

1. Принцип обучения конструированию программ на подробно комментированных образцах решения тщательно подобранных задач. Назначение примеров — не только дать образцы и описать основные схемы алгоритмов, но и на сравнительном анализе разных решений одной и той же задачи

познакомить студента с такими понятиями, как эффективность, наглядность и надежность решения.

2. Принцип доказательного программирования, когда программа строится вместе с доказательством ее правильности. Для этого в курсе вводятся понятия промежуточных утверждений и инвариантов, а в разрабатываемых алгоритмах решения задач такие утверждения записываются в форме программных комментариев.

3. Принцип пошаговой разработки программ, когда программа строится из формальной спецификации задачи с помощью мелких формально проверяемых шагов преобразования.

4. Принцип модульного программирования, позволяющий проектировать, разрабатывать и собирать программу по частям и с использованием библиотек уже готовых частей.

5. Принцип объектно-ориентированного программирования, позволяющий разработчикам программ легко создавать все более сложные приложения с помощью инкапсуляции, наследования и полиморфизма.

4.3. Введение в программирование

Книга [20] предназначена для изучения студентами основных понятий программирования и постепенного овладения навыками разработки на языке Zonnon линейных, ветвящихся и итеративных алгоритмов, алгоритмов с процедурами и со структурированными данными.

Книга содержит порядка 3000 задач и состоит из 6 глав.

Внутри одной главы задачи сгруппированы по методам решения и языковым конструкциям, на освоение которых они ориентированы. Каждая группа составляет самостоятельный параграф, содержащий помимо текстов задач либо описание соответствующих языковых понятий, либо примеры решения некоторых типичных задач данной группы. Назначение примеров — не только дать образцы и описать основные схемы алгоритмов, но и показать, как алгоритм может быть выведен из рекурсивных соотношений — спецификации алгоритма; привести инварианты циклов и другие промежуточные утверждения, из которых выводится правильность программы. Здесь же на сравнительном анализе разных решений одной и той же задачи студент знакомится с такими понятиями, как эффективность, наглядность и надежность решения.

Главу завершает список заданий. Каждая из формулировок заданий представляет собой фактически схему для построения конкретных вариантов задания — индивидуальных задач на программирование для всех сту-

дентов учебной группы. Эти варианты получаются в результате подстановки в текст вместо номера варианта его значения и выбора по значению номера варианта подходящих частей текста, составляющего формулировку задания.

Например, 7-ой вариант следующего задания:

«В заданной последовательности целых чисел найти
номер (при N **mod** 3 = 0),
модуль (при N **mod** 3 = 1),
квадрат (при N **mod** 3 = 2)
такого
первого (при N **mod** 4 = 0),
последнего (при N **mod** 4 = 1),
минимального (при N **mod** 4 = 2),
максимального (при N **mod** 4 = 3)
элемента, который является
четным (при N **mod** 2 = 0),
не кратным N (при N **mod** 2 = 1)
числом и совпадает с кодом некоторой
буквы (при N **mod** 8 > 3),
цифры (при N **mod** 8 ≤ 3).»

будет иметь вид:

«В заданной последовательности целых чисел найти модуль такого максимального элемента, который является не кратным 7 числом и совпадает с кодом некоторой буквы.»

Гл. 1 содержит вводные понятия и начинается с ответов на такие вопросы, связанные с алгоритмами и функциями, как: что такое алгоритм, что такое компьютер, как можно определять функции и как можно определить язык Zonnon?

Методам построения простейших программ посвящена гл. 2. В ней рассматриваются стандартные типы данных и средства организации вычислений в линейных программах. Излагаются вопросы доказательства свойств программ: поясняется, зачем нужны доказательства правильности программ, вводятся понятия промежуточных утверждений, внешней спецификации программы, полной и частичной правильности программ, описываются методы задания внешней спецификации простых программ, и излагается метод промежуточных утверждений доказательства их свойств. Приводятся образцы разработки линейных программ на примере решения таких задач, как периметр и площадь прямоугольного треугольника, симметричная буква, возведение в степень и площадь треугольника.

Гл. 3 посвящена методам построения простых программ без циклов. Она начинается с описания средств для организации и описания ветвлений в программах: вводятся блок-схемные представления программ и их фрагментов, описываются условные операторы и операторы выбора, излагаются методы доказательства свойств ветвящихся программ. Затем приводятся образцы разработки ветвящихся программ на примере решения таких задач, как точка в треугольнике, максимум из трех чисел, табличное задание функции, анализ квадратного уравнения и определение типа треугольника.

Гл. 4 посвящена методам построения итеративных программ. В ней описываются средства для организации и анализа свойств циклических вычислений: операторы цикла с условием на продолжение, операторы цикла с условием на окончание и цикла с параметрами. Излагаются методы пошаговой разработки программ. Приводятся образцы решения задач независимой обработки элементов последовательности (перекодировщик, выборка элементов последовательности), вычисления элементов последовательности (нахождение факториала, вычисление числа e, схема Горнера), реализации функций на последовательностях (подсчет вхождений, количество максимумов), обработки последовательности последовательностей и реализации двойных циклов (обработка слов предложения, приближенное вычисление сумм рядов).

Вопросам построения программ обработки структурированных данных посвящена гл. 5. В ней описываются средства задания и анализа программ со структурами данных, приводятся образцы решения задач обработки векторов (векторная функция, поиск в упорядоченном векторе, подсчет количества вхождений каждой буквы, свертка вектора) и программы обработки матриц (поиск максимума в матрице, произведение матриц, преобразование матриц, поиск номера строки-серии).

Гл. 6 посвящена классу программ с процедурами и функциями. В ней рассматриваются правила описания и вызовов процедур, блоки и локализация имен, изменение действий (входные параметры) и получение результатов (выходные параметры) процедур, а также вычисление единственного значения (функции). Излагаются методы использования процедур и функций для пошаговой разработки программ, рассматриваются рекурсивные подпрограммы и исследуется, когда и как нужно использовать рекурсию, описывается метод структурной индукции для доказательства свойств рекурсивных программ. Приводятся образцы решения задач с использованием процедур и функций, таких как обработка последовательности векторов, Ханойские башни, кратные суммы, быстрая сортировка и числа Фибоначчи.

4.4. Цели и содержание практикума

Основной задачей компьютерного практикума является не только и не столько обучение студентов собственно записи (кодированию) известного алгоритма на языке Zonnon для платформы .NET, а практическое закрепление знаний, получаемых в ходе лекций и семинарских занятий по вводному курсу программирования, и овладение общими методами, приемами и навыками технологии решения задач на компьютере. Основная цель, которая ставится перед студентом при выполнении индивидуальных заданий, составляющих компьютерный практикум, — это практическое освоение всех этапов разработки надежной и наглядной интерактивной (диалоговой) Zonnon-программы для компьютерного решения несложной задачи, требующей разработки алгоритма, обработки сложных структур данных и создания дружественного интерфейса.

В силу этого тематика индивидуальных заданий для компьютерного практикума определяется, в первую очередь, всеми видами работ, которые должен освоить студент, чтобы научиться создавать качественные (эффективные, наглядные и надежные) нетривиальные программы. В большинстве случаев задача, решаемая во время выполнения индивидуального задания, — это задача невычислительного характера, имеющая краткую и точную (содержательную) формулировку и допускающая большое разнообразие решений.

Вместе с тем студенту и преподавателю нужно иметь в виду, что многие из методов и понятий, используемых в индивидуальных заданиях, являются неотъемлемой частью образования современного специалиста, использующего компьютер для решения своих задач. Поэтому предполагается, что при прохождении компьютерного практикума студент получит не менее пяти индивидуальных заданий, по одному из каждого тематического раздела.

Выполнение каждого задания, составляющего компьютерный практикум, в общем случае включает следующие виды работ:

- анализ условия задачи и выработка подхода к ее решению;
- пошаговая разработка (на основании выбранного подхода) алгоритма решения и его описание;
- обоснование алгоритма;
- выбор и обоснование представления для входных, выходных и промежуточных данных;
- кодирование алгоритма, т.е. его запись на языке Zonnon;

- выбор и обоснование набора тестов, на которых будет проверяться программа;
- отладка программы и демонстрация ее правильной работы на выбранном наборе тестов.

Это разбиение условно в том смысле, что фактически некоторые виды работ тесно переплетаются и выполнение их обычно составляет единый процесс. Например, строить набор тестов удобнее одновременно с построением самого алгоритма, а обосновывать правильность работы алгоритма удобно путем детальной демонстрации процесса его построения.

Выполнение каждого задания студент завершает составлением отчета, который включает:

- формулировку задачи;
- описание программы для пользователя, ее внешнюю спецификацию, т.е. описание способа задания входных данных, вида результатов программы при заданных входных данных и сценария диалога в процессе исполнения программы;
- словесное описание алгоритма и обоснование его правильности и эффективности;
- текст программы;
- описание тестового набора и его обоснование.

Предполагается, что студент, решая задачу, создает интерактивную (диалоговую) Zonnon программу для платформы .NET, которая за один запуск может обрабатывать не один входной набор, а последовательность входных наборов произвольной длины. Эти входные наборы либо заранее размещаются в специальных «входных» файлах программы, предъявляемых преподавателю вместе с разработанной программой, либо задаются пользователем программы в процессе ее исполнения. Таким образом, разработанная студентом программа после завершения обработки очередного входного набора в зависимости от указания пользователя может либо завершить свою работу (остановиться), либо продолжить ее и перейти к обработке следующего входного набора. В последнем случае программа вступает в диалог с пользователем, в процессе которого пользователь программы готовит и инициирует новый счет по программе. Для этого он либо вводит очередной входной набор с помощью клавиатуры и мышки, либо дает указание программе, из какого файла ей нужно взять необходимые данные.

При этом совсем не требуется, чтобы при переходе программы к обработке следующего входного набора этот набор каждый раз целиком задавался заново. Например, во многих заданиях удобно разделить входной

набор на две части, выделяя более общие (как правило, более объемные) исходные данные задачи в отдельную часть, и предусмотреть специальный вид реализации счета по программе на последовательности входных данных, различающихся второй частью, без необходимости повторного задания первой. Например, в задаче нахождения кратчайшего пути между двумя заданными городами заданной системы дорог более общей (и более объемной) частью входного набора является система дорог. Поэтому естественно решение, при котором система дорог является общей для нескольких следующих друг за другом счетов по программе. В этом случае каждый раз, переходя к новому счету по программе, пользователь может выбрать один из двух вариантов продолжения работы:

- осуществить ввод очередной системы дорог из заданного файла с выводом его изображения на экран;
- запустить нахождение кратчайшего пути между двумя заданными городами в текущей системе дорог.

Важно, чтобы до выполнения заданий студент и преподаватель согласовали все требования, предъявляемые к их выполнению, в том числе, виды работ по каждому заданию, а также порядок и сроки сдачи заданий. В любом случае понимание условий задач студентом не должно отличаться от понимания преподавателем. Поэтому раньше, чем студент начнет разрабатывать алгоритм, студент и преподаватель должны вместе тщательно проанализировать условие задачи и обсудить особенности и трудности ее решения, зафиксировать интерфейс программы, включая представление входных и выходных данных.

Необходим также постоянный контроль текущего состояния решения заданий для координации установленных преподавателем требований и действительных намерений студента. Необходимо уделять особое внимание высокой надежности создаваемых программ. Программа должна содержать достаточное число так называемых стопоров ошибок — динамических проверок справедливости утверждений, характеризующих правильность функционирования программы и ее применения. Очень важно, чтобы программа не только давала правильные результаты для корректных исходных данных, но и осмысленно реагировала на некорректные данные и указания пользователя. Не менее важно, чтобы программа выдавала на экран (или в специальный файл) в удобном виде информацию о ходе вычисления по программе в таком объеме (и в таком виде), который позволяет легко идентифицировать ошибку (как в программе, так и во входных данных) и локализовать место ее возникновения.

Программа обязательно должна быть наглядной, т.е. хорошо структурированной, с достаточным количеством комментариев и т.д., а также содержать инструкцию по своему использованию («встроенный help»).

Особое внимание нужно уделить правильности программы и полноте тестового набора. Описание каждого теста помимо файла с входными данными должно включать информацию, достаточную для оценки правильности работы программы на этих входных данных. Минимальное требование к тестовому набору состоит в том, что каждый оператор программы должен быть достижим (т.е. выполняться) хотя бы на одном тесте из этого набора. Поэтому среди тестов набора должны быть представлены и некорректные исходные данные.

Входные и выходные данные разработанной программы должны быть естественными для человека. Степень естественности представления определяется объемом той работы, которая требуется для перехода от содержательного описания входных данных к их представлению (очень часто неестественность входного представления проявляется в его неоправданно большом размере) и от представления выходных данных к их содержательному пониманию.

Задачи, составляющие индивидуальные задания, допускают, как правило, целый спектр различных по эффективности решений. Предполагается, что студент должен выбрать и обосновать по возможности хорошее решение. Минимальным требованием к эффективности решения является успешная работа программы на согласованном с преподавателем тестовом наборе.

Для удобства все индивидуальные задания разбиты на три группы:

- обычной сложности (их формулировки не имеют пометки),
- повышенной сложности (с пометкой “↑”) и
- пониженной сложности (с пометкой “↓”).

При оценке сложности задания рассматривались три показателя:

- сложность структур данных,
- сложность вычислений и
- изобретательность.

В показатель «изобретательность» включались такие свойства задания, как непривычность для студента понятий, используемых в задании, сложность извлечения из определений тех свойств, на которых должен базироваться алгоритм решения задания, а также сложная связь между структурами данных и вычислениями. Каждый из указанных трех показателей задания оценивался в баллах 0, 1 или 2. Пометку “↓” получили задания, на-

бравшие в сумме по трем показателям всего 1 балл, а пометку "↑"— задания, суммарный балл которых больше 3.

4.5. Практикум по программированию

Книга [23] предназначена для проведения компьютерного практикума по курсу программирования. Она содержит 500 индивидуальных заданий различной сложности, ориентированных на приобретение студентами навыка практического решения задач создания пользовательских приложений и Web-приложений для платформы Microsoft.NET, требующих разработки алгоритма, обработки сложных структур данных и разработки дружественного интерфейса. Каждое индивидуальное задание — это самостоятельная, как правило, комбинаторная или логическая задача с краткой и четкой формулировкой, не содержащей описания алгоритма. Тематические задачи разбиты на пять разделов по 100 заданий в каждом разделе:

- графы и системы дорог;
- грамматики, языки и автоматы;
- формулы и программы;
- геометрия;
- игры и модели.

Основную часть книги составляют подробные рекомендации по выполнению разных видов работ, которые должен освоить студент, чтобы научиться создавать эффективные, наглядные и надежные нетривиальные приложения для платформы Microsoft.NET, в том числе для работы в среде Интернет.

Книга состоит из введения и 6 глав.

Нужно сказать, что при составлении сборника было стремление сделать изложение доступным для читателей, не обладающих специальными знаниями ни по математике, ни по программированию. Поэтому все необходимые для решения задач сведения по программированию, в том числе и полное описание языка Zonnon (гл. 1), включены в сборник, а условия большинства заданий и упражнений доступны для учащихся старших классов. Приведенные в сборнике (гл. 2) определения понятий, встречающихся в заданиях, вполне достаточны для того, чтобы можно было понимать формулировки заданий (гл. 6). Однако для построения эффективного решения некоторых заданий может оказаться полезным знание некоторых свойств определяемых здесь математических объектов. Поэтому во введении не только описываются цели и задачи практикума и даются общие рекоменда-

ции преподавателю и студенту, но и дается обзор дополнительной литературы, которая может помочь студенту при выполнении его заданий.

Гл. 3, 4 и 5 содержат подробные рекомендации по решению индивидуальных заданий, составляющих практикум. В гл. 3 даются общие рекомендации по разработке и обоснованию алгоритма, представлению данных, анализу алгоритма и его сложности, выбору и обоснованию тестов, начиная с анализа условия задачи и кончая отладкой созданной программы. Особенностью заданий данного практикума является то, что они связаны со сложными структурами данных, с организацией поиска или порождения тех элементов конечного (но, как правило, чрезвычайно большого) множества, которые удовлетворяют определенным ограничениям, а также то, что требуют при своем решении анализа алгоритма и его сложности. В гл. 4 приводятся рекомендации по алгоритмическому решению задач, связанных с организацией вычислений на дискретных конечных математических структурах. В ней описываются основные типы данных, возникающие при выполнении заданий (списки, стеки, очереди, деревья, графы, орграфы), приводится язык описания алгоритмов, рассматривается поиск с возвращением, описываются алгоритмы поиска с возвращением, обходов ордерева в глубину и в ширину, обходов графа в глубину и в ширину, дается усовершенствование поиска с возвращением, приводятся методы ветвей и границ, динамического программирования, а также порождения объектов, перестановок, подмножеств множества, сочетаний. В гл. 5 рассматриваются вопросы выбора представления для структур данных, присутствующих в формулировках заданий, и даются конкретные рекомендации по выбору представлений для таких структур данных, как множество точек на плоскости, прямая на плоскости, окружность и многоугольник, последовательность, вектор, матрица, тексты, слова и предложения, граф, корневое дерево, система дорог, грамматика, автоматная диаграмма, логическая формула, логический фрагмент, простое выражение, полином, игры.

4.6. Вопросы использования курса в рамках системы WAPE

Нетрудно увидеть, что структура и содержание курса напрямую соответствуют требованиям системы WAPE. В первую очередь это — разделение учебного материала на два учебных пособия, а также большой набор (порядка 3 тыс.) задач и их разделение на упражнения и задания.

Набор упражнений содержит достаточно задач по каждой теме, чтобы обеспечить процесс тестирования.

В учебных пособиях имеется достаточный широкий набор примеров решения задач.

Задания семинарских занятий легко формулируются на языке ЯЗП для их ввода в качестве проектов. Например, задание, приведенное в разд.4.3., можно записать на языке ЯЗП следующим образом:

```
## В заданной последовательности целых чисел найти
#<# N MOD 3, 0> номер
#<# N MOD 3, 1> модуль
#<# N MOD 3, 2> квадрат
##такого
#<# N MOD 4, 0> первого
#<# N MOD 4, 1> последнего
#<# N MOD 4, 2> минимального
#<# N MOD 4, 3> максимального
##элемента, который является
#<# N MOD 2, 0> четным
#<# N MOD 2, 1> не кратным #N
##числом и совпадает с кодом некоторой
#<# N MOD 8, (3, 7]> буквы.
#<# N MOD 8, [0, 3]> цифры.
```

Выводы по главе 4

В главе представлен вводный курс по программированию на базе языка Zonnon, работа над которым ведется в Цюрихском институте информатики. Курс опирается на опыт преподавания основного курса по программированию для студентов механико-математического факультета НГУ с использованием языка Паскаль и легко встраивается в систему WAPE. Учебные пособия «Введение в программирование» и «Практикум по программированию», поддерживающие курс, размещены на сайте русскоязычной библиотеки учебных курсов международной программы MSDN Academic Alliance [53] и уже более двух лет доступны для использования всем желающим, имеющим выход в Интернет. Есть заинтересованность виртуальных университетов к размещению курса на своих сайтах.

Язык Zonnon задуман как дальнейшая эволюция хорошо известного и широко применяемого на западе в учебных целях языка Оберон, являющегося преемником языков Паскаль и Модула-2. Развивая язык Оберон, исхо-

дя из современных потребностей в программировании, авторы сохранили в языке Zonnon такие важные черты Оберона и его предшественников, как компактность языка, ясность, недвусмысленность и ортогональность его основных понятий. Поэтому можно ожидать, что язык Zonnon и данный вводный курс будут востребованы теми учебными заведениями, которые в настоящее время используют Паскаль в качестве языка начального обучения программированию и имеют желание перейти к более современному курсу программирования, охватывающему концепции языков программирования нового поколения, таких как Java и C#, но осуществить этот переход плавно, без резкого изменения сложившегося стиля преподавания программирования.

И хотя пока еще нет практики использования языка Zonnon для курсов по современному программированию, несомненно то, что возможности языка Zonnon позволяют разработать на его основе полноценные курсы как по объектно-ориентированному программированию, так и по параллельному программированию. При наличии таких курсов студент сможет овладевать основами программирования и современными методами программирования в рамках одного языка, причем языка, который ориентирован на поддержку обучения и удобен для обучения.

СПИСОК ЛИТЕРАТУРЫ

1. Атанов Г.А. Деятельностный подход в обучении. — Донецк: ЕАИ-пресс, 2000.
2. Атанов Г.А. Методика и методология проблемного обучения. — Донецк: Изд-во ДонГУ, 1990.
3. Бертон В. А. Принципы обучения и его организация. — М.: Учпедгиз, 1934.
4. Бочаров А.О. Система обучения программированию: подсистема тестирования // Курсовая работа. — Новосибирск: НГУ, 2003.
5. Брушлинский А.В. Психология мышления и проблемное обучение. — М.: Педагогика, 1983.
6. Бутин К.А. Система обучения программированию: подсистема решения заданий // Выпускная квалификационная работа бакалавра. — Новосибирск: НГУ, 2003.
7. Волянская Т.А. Виртуальный музей истории информатики в Сибири: модель предметной области и модель пользователя // Новые информационные технологии в науке и образовании / Под ред. В.Н. Касьянова. — Новосибирск: ИСИ СО РАН, 2003. — С. 124–146.
8. Волянская Т.А. Методы и технологии адаптивной гипермедиа // Современные проблемы конструирования программ / Под ред. В.Н. Касьянова. — Новосибирск: ИСИ СО РАН, 2002. — С. 38–68.
9. Гальперин П.Я. Основные результаты исследования по проблеме «Формирование умственных действий и понятий». — М.: Педагогика, 1965.
10. Джонсон Б., Скибо К., Янг М. Основы Microsoft Visual Studio .NET 2003. — М.: Русская Редакция, 2003.
11. Диалоговые системы. Современное состояние и перспективы развития / Довгялло А.М., Брановицкий В.И., Вершинин К.П. и др. — Киев: Наукава думка, 1987. — 248 С.
12. Евстигнеев В.А., Касьянов В.Н. Толковый словарь по теории графов в информатике и программировании. — Новосибирск: Наука, 1999. — 288 с.
13. Зайцева Л.В. Методы и модели адаптации к учащимся в системах компьютерного обучения // Educational Technology & Society. — 2003. — Vol. 6, N 4. — P. 204–211.
14. ИнтУИТ-университет . — http://www.intuit.ru/
15. Информационные технологии в науке и образовании: Международная науч.-практ. конф. МКИТО-2001. — Шахты: Изд-во ЮРГУЭС, 2001.
16. Касьянов В.Н. Курс программирования на Паскале в заданиях и упражнениях. — Новосибирск: НГУ, 2001.
17. Касьянов В.Н. О работе 16 Всемирного компьютерного конгресса ИФИП // Поддержка супервычислений и интернет-ориентированные технологии / Под ред. В.Н. Касьянова. — Новосибирск: ИСИ СО РАН, 2001.
18. Касьянов В.Н. Язык программирования Минал. — Новосибирск: НГУ, 1979.

19. Касьянов В.Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение. — СПб.: БХВ-Петербург, 2003. — 1104 с.
20. Касьянов В.Н., Касьянова Е.В. Введение в программирование. — 2004. — 250 с. — http://www.microsoft.com/Rus/Msdnaa/Curricula/Default.mspx
21. Касьянов В.Н., Касьянова Е.В. Дистанционное обучение: методы и средства адаптивной гипермедиа // Программные средства и математические основы информатики / Под ред. В.Н. Касьянова. — Новосибирск, 2004. — С. 80—141.
22. Касьянов В.Н., Касьянова Е.В. Дистанционное обучение: методы и средства адаптивной гипермедиа // Вычислительные технологии. — 2004. — Т.9, Часть 2. — С. 333–341. — (Специальный выпуск по материалам Междунар. конф. ВИТ).
23. Касьянов В.Н., Касьянова Е.В. Практикум по программированию. —2004. — 200 с. — http://www.microsoft.com/Rus/Msdnaa/Curricula/Default.mspx
24. Касьянов В.Н, Касьянова Е.В. Адаптивные системы и методы дистанционного обучения // Информационные технологии в высшем образовании. — 2004.— Т. 1, № 4. — С. 40–60.
25. Касьянова Е.В. Освоение технологии программирования ребятами старших классов в процессе выполнения проекта во время летней практики // Тез. докл. II-го Всесибирского конгресса женщин-математиков. — Красноярск: КГУ, 2002. — С. 92–93.
26. Касьянова Е.В. Освоение технологии программирования ребятами старших классов в процессе выполнения проекта во время летней практики // Тр. II-го Всесибирского конгресса женщин-математиков. — Красноярск: КГУ, 2002. — С. 58–61.
27. Касьянова Е.В. Язык программирования Zonnon для платформы .NET // Программные средства и математические основы информатики / Под ред. В.Н. Касьянова. — Новосибирск, 2004. — С. 189–205.
28. Касьянова Е.В. Вводный курс программирования на базе языка Zonnon // Методы и инструменты конструирования и оптимизации программ / Под ред. В.Н. Касьянова. — Новосибирск, 2005. — С. 95–116.
29. Касьянова Е.В. Методические и программные средства поддержки дистанционного обучения программированию // Технология Microsoft в теории и практике программирования. Тез. конференции-конкурса. — Новосибирск: НГУ, 2006. — С. 89–91.
30. Касьянова Е.В. WAPE — адаптивная система поддержки дистанционного обучения программированию // Тр. Междунар. конф. «Вычислительные и информационные технологии в науке и образовании». — Павлодар, 2006. — Т. 1. — С. 606–615.
31. Касьянова Е.В. Адаптивная система поддержки дистанционного обучения программированию // Проблемы интеллектуализации и качества систем информатики / Под ред. В.Н. Касьянова. — Новосибирск: ИСИ СО РАН, 2006. — С. 85–112.

32. Касьянова Е.В. Адаптивные методы и средства дистанционного обучения // Современные ценности и эффективность моделей образовательных систем. Материалы Междунар. научно-практической конф. — Новосибирск: Изд-во НИПКиПРО, 2006. — Часть 2. — С. 97–100.
33. Касьянова Е.В. Адаптивные методы и средства поддержки дистанционного обучения программированию // VI Междунар. конф. памяти А.П. Ершова «Перспективы систем информатики». Секция «Информатика образования». Доклады и тезисы. — Новосибирск, 2006. — С. 28–31.
34. Касьянова Е.В. Адаптивные технологии дистанционного обучения программированию // Современные проблемы науки и образования. — 2006. — № 2. — С. 92–93.
35. Касьянова Е.В. Адаптивная система поддержки дистанционного обучения программированию: вводный курс и методы тестирования // Технологии Microsoft в информатике и программировании. — Новосибирск, 2007. — С. 55–57.
36. Касьянова Е.В. Моделирование знаний студента в адаптивной системе дистанционного обучения // Технологии Microsoft в информатике и программировании. — Новосибирск, 2007. — С. 57–59.
37. Касьянова Е.В., Касьянова С.Н. Программирование для школьников: сборник задач повышенной сложности с решениями. — Новосибирск, 2002. — 50 с. — (Препр. / РАН. Сиб. Отд-е. ИСИ; № 95).
38. Касьянова Е.В., Касьянова С.Н. Опыт преподавания информатики в старших классах с математическим уклоном // Современные проблемы науки и образования. — 2006. — N 2. — С. 91– 92.
39. Касьянова С.Н., Касьянова Е.В. Опыт преподавания программирования в старших классах // V Междунар. конф. памяти А.П. Ершова «Перспективы систем информатики». Секция «Информатика образования». Доклады и тезисы. — Новосибирск, 2003. — С. 31.
40. Компьютерное общество IEEE. — http://www.ieee.com/portal/site/iportals/
41. Лефевр В.А. Конфликтирующие системы. — М, 1973.
42. Махмутов М. И. Проблемное обучение. — М.: Педагогика, 1975.
43. Новые информационные технологии в региональной инфраструктуре и образовании: Материалы IV Междунар. науч.-практ. конф. — Астрахань: АГТУ, 2001.
44. Образование в конце XX века (материалы «круглого стола»). Выступили: Зотов А.Ф., Купцов В.И., Розин В.М., Марков А.Р., Шикин Е.В., Царев В.Г., Огурцов А.П. // Вопросы философии. — 1992. — № 9.
45. Оконь В. Основы проблемного обучения. — М.: Просвещение, 1968.
46. Оптнер С.Л. Системный анализ для решения деловых и промышленных проблем. — М.: Мир, 1969.
47. Основы языка ПАСКАЛЬ-360 / Сост. В.Н. Касьянов. — Новосибирск: НГУ, 1982.

48. Попов Э.В., Фирдман Г.Р. Алгоритмические основы интеллектуальных роботов и искусственного интеллекта. — М.: Наука, 1976.
49. Поспелов Г.С. Искусственный интеллект — основа новой информационной технологии. — М.: Наука, 1988.
50. Поспелов Д.А. Моделирование рассуждений. — М.: Радио и связь, 1989.
51. Программирование на мини-Фортране / Сост. В.Н. Касьянов. — Новосибирск: НГУ, 1981.
52. Просиз Дж. Программирование для .NET. — М.: Русская Редакция, 2003.
53. Сайт русскоязычной библиотеки учебных курсов международной программы MSDN Academic Alliance. — http://www.microsoft.com/Rus/Msdnaa/Curricula/Default.msp
54. Технологии информационного общества — Интернет и современное общество: Материалы Всероссийской объединенной конф. — Санкт-Петербург: СПбГУ, 2001.
55. Уоткинз Д., Хаммонд М., Эйбрамз Б. Программирование на платформе .NET. — М.: Вильямс, 2003.
56. Щедровицкий Г.П. Процессы и структуры в мышлении. — М.: Путь, 2003.
57. Элти Дж., Кумбс М. Экспертные системы: концепции и примеры. — М.: Финансы и статистика, 1987.
58. Albrecht D., Zukerman I., Nicholson A., Bud A. Towards a Bayesian model for keyhole plan recognition in large domains // Proc. of the 6th Internat. Conf. on User Modeling (UM97). — Sardinia, 1997.
59. Alpert S.R., Singley M.K., Fairweather P.G. Deploying intelligent tutors on the Web: An architecture and an example // Intern. J. of Artificial Intelligence in Education. — 1999. —Vol. 10. — P. 183–197.
60. Baralis E., Widom J. An algebraic approach to static analysis of active database rules // ACM Transactions on Database Systems. — 2000. — Vol. 20, N 1. — P. 269–332.
61. Beaumont J. User modelling in the interactive anatomy tutoring system ANATOM-Tutor // User Modeling and User Adapted Interaction. — 1994. — Vol. 4, N 1. — P. 21–45.
62. Berners-Lee T. World Wide Web: An illustrated seminar. Held as an Online Seminar in 1991. — http://www.w3.org/pub/WWW/Talks/General.html.
63. Brusilovsky P. Adaptive and intelligent technologies for Web-based education // Konstliche Intelligenz. Special Issue on Intelligent Systems and Teleteaching. — 1999. — N 4. — P. 19–25.
64. Brusilovsky P. Adaptive educational systems on the World-Wide-Web: a review of available technologies // Proc. of Workshop "WWW-Based Tutoring" at the 4th Intern. Conf. on Intelligent Tutoring Systems (ITS'98). — San Antonio, 1998.
65. Brusilovsky P. Adaptive hypermedia // User Modeling and User-Adapted Interaction. — 2001. — Vol. 11. —P. 87–110.
66. Brusilovsky P. Adaptive hypermedia, an attempt to analyze and generalize. // Lect. Notes Comput. Sci. — 1996. — Vol. 1077. — P. 288–304.

67. Brusilovsky P. Efficient techniques for adaptive hypermedia // Lect. Notes Comput. Sci. — 1997. — Vol. 1326 — P. 12–30.
68. Brusilovsky P. Methods and techniques of adaptive hypermedia // User Modeling and User-Adapted Interaction. — 1996. — Vol. 6, N 2-3. — P. 87–129.
69. Brusilovsky P., Cooper D. W. Domain, task, and user models for an adaptive hypermedia performance support system. // Proc. of the 2002 Internat. Conf. on Intelligent User Interfaces. — San Francisco, CA, 2002. — P. 23–30.
70. Brusilovsky P., Pesin L. ISIS-Tutor: An intelligent learning environment for CDS/ISIS users // Proc. of the Interdisciplinary Workshop on Complex Learning in Computer Environments (CLCE'94). — Joensuu, 1994.
71. Brusilovsky P., Schwarz E. User as student: Towards an adaptive interface for advanced web-based applications // Proc. of the 6th Internat. Conf. on User Modeling (UM97). — Sardinia, 1997. — P. 112–117.
72. Brusilovsky P., Schwarz E., Weber G. A tool for developing adaptive electronic textbooks on WWW // Proc. of World Conf. of the Web Society (WebNet'96). — Boston, 1996. — P. 64–69.
73. Brusilovsky P., Schwarz E., Weber G. ELM-ART: An intelligent tutoring system on world wide web // Lect. Notes Comput. Sci. — 1996. — Vol. 1086. — P. 261–269.
74. Calvi L., De Bra P. Improving the usability of hypertext courseware through adaptive linking // The 8th ACM Internat. Hypertext Conf. — Southampton, 1997. — P. 24–29.
75. Carver C.A., Howard R.A., Lavelle E. Enhancing student learning by incorporating student learning styles into adaptive hypermedia // Proc. of World Conf. on Educational Multimedia and Hypermedia (ED-EDIA'96), — Boston, MA, 1996. — P. 118–123.
76. Carver C.A., Howard R.A., Lavelle E. Enhancing student learning by incorporating student learning styles into adaptive hypermedia // Proc. of World Conf. on Educational Multimedia and Hypermedia (ED-MEDIA'96). — Boston, 1996. — P. 118–123.
77. Chan A.T.S., Chan S.Y.C., Cao J. SAC: a self-paced and adaptive courseware systems // IEEE Intern. Conf. on Advanced Learning Technologies. — IEEE Computer Society, 2001. — P. 78—81.
78. Conati C., Gertner A.S., Van Lehn K., and Druzdzel M. J. Online student modeling for coached problem solving using Bayesian networks // Proc. of the 6th Internat. Conf. on User Modeling (UM97). — Sardinia, 1997.
79. Cooper G. The computational complexity of probabilistic inference using Bayesian belief networks // Artificial Intelligence. — 1990. — Vol. 42. — P. 393–405.
80. Cowell R.G., Dawid A.P., Lauritzen S.L., Spiegelhalter D.J. Probabilistic Networks and Expert Systems. — Springer, 1999.
81. Dagum P., Galper A., Horvitz, E. Dynamic network models for forecasting // The 8th Conf. on Uncertainty in Artificial Intelligence. — San Mateo, 1992. — P. 41–48.

82. Danielson R. Learning styles, media preferences, and adaptive education // Proc. of Workshop "Adaptive Systems and User Modeling on the World Wide Web" at the 6th Internat. Conf. on User Modeling (UM97) — Sardinia, 1997. — P. 31–35.
83. De Bra P. Adaptive hypermedia on the Web: Methods, techniques and applications // Proc. of the AACE WebNet'98 Conf. — Orlando, 1998. — P. 220–225.
84. De Bra P. Design issues in adaptive Web-site development // Proc. of the 2nd Workshop on Active Systems and User Modelling on WWW. — Eindhoven, 1999. — P. 29–39.
85. De Bra P. Hypermedia structures and systems: Online Course at Eindhoven University of Technology, 1997. — http://wwwis.win.tue.nl/2L690/.
86. De Bra P. Teaching hypertext and hypermedia through the Web // Proc. of of the Web Society (WebNet'96). — San Francisco, 1996. — P.130–135.
87. De Bra P., Aerts A., Smits D., Stash N. AHA! Version 2.0, more adaptation flexibility for authors // Proc. of the AACE ELearn'2002 Conf. — 2002. — P. 240–246.
88. De Bra P., Aerts A., Smits D., Stash N. AHA! Version 2.0, More Adaptation Flexibility for Authors // Proc. of the AACE ELearn'2002 Conf. — 2002. — P. 240–246.
89. De Bra P., Brusilovsky P., Houben G.-J. Adaptive hypermedia: from systems to framework // ACM Computing Surveys. — 1999. — Vol. 31, N 4. — Article N 12.
90. De Bra P., Calvi L. AHA! An open adaptive hypermedia architecture // The New Review of Hypermedia and Multimedia. — 1998. — Vol. 4. — P. 115–139.
91. De Bra P., Houben G.J., Wu H., AHAM: A Dexter-based reference model for adaptive hypermedia // Proc. of ACM Hypertext'99. — Darmstadt, 1999. — P. 147–156.
92. De Bra P., Stash N. AHA! A general-purpose tool for adaptive websites // Lect. Notes Comput. Sci. — 2002. — Vol. 2347. — P. 381–384.
93. Dechter R. Enhancement schemes for constraint processing: Back jumping, learning and cutset decomposition // Artificial Intelligence 1989. — Vol.41. — P.273-312.
94. Dechter R., and Pearl J. Tree clustering for constraint networks // Artificial Intelligence. — 1989. — Vol.38. — P. 353-366.
95. Desmarais M.C., Maluf A. User-expertise modeling with empirically derived probabilistic implication networks // User Modeling and User Adapted Interaction. — 1996. — Vol. 5. — P. 283–315.
96. Fink J., Kobsa A., Schreck J. Personalized hypermedia information provision through adaptive and adaptable systems features: User modeling, privacy and security issues // Intelligence in Services and Networks: Technology for Cooperative Competition. — Springer-Verlag, 1997. — P. 459–467.
97. Gilbert J. E., Han C.Y. Arthur: Adapting Instruction to Accommodate Learning Style // Proc. of World Conf. of the WWW and Internet (WebNet'99). — Honolulu: HI, 1999. — P. 433–438.
98. Goldstein I. The genetic graph: A representation for the evolution of procedural knowledge // Intelligent Tutoring Systems. — Academic Press, 1982.
99. Greer J., McCalla G., Collins J., Kumar V., Meagher P., Vassileva J. Supporting peer help and collaboration in distributed workplace environments // Intern. J. of Artificial Intelligence in Education. — 1998. — Vol. 9. — P.159–177.

100. Gronbaek K., Trigg R.H. From Web to Workplace: Designing Open Hypermedia Systems. — The MIT Press, 1999.
101. Gutknecht J., Zueff E. Zonnon language experiment, or how to implement a non-conventional object model for .NET // OOPSLA'02. — Seattle, Washington, 2002.
102. Gutknecht J., Zueff E. Zonnon Language Report. — Zurich, Institute of Computer Systems ETH Zentrum, 2004.
103. Halasz F., Schwartz M. The Dexter hypertext reference model // Communications of the ACM. — 1994. — Vol. 37, N 2. — P. 30–39.
104. Henze N., Nejdl W. Adaptivity in the KBS hyperbook system // The 2nd Workshop on Adaptive Systems and User Modeling on the WWW. — Toronto, 1999.
105. Henze N., Nejdl W., Wolpers M. Modeling constructivist teaching functionality and structure in the KBS hyperbook system // CSCL'99: Computer Supported Collaborative Learning. — Standford, 1999.
106. Hook K., Karlgren J., Waern A., Dahlback N., Jansson C., Karlgren K., Lemaire B. A glass box approach to adaptive hypermedia // User Modeling and User Adapted Interaction. — 1996. — Vol. 6, N 2-3. — P. 157–184.
107. Hoppe U. Use of multiple student modeling to parameterize group learning // Proc. of VII World Conf. on Artificial Intelligence in Education (AI-ED'95). — Washington, DC, 1995. — P. 234–249.
108. Ito J., Okazaki Y., Watanabe K., Kondo H., Okamoto M.: Pen based user interface for an ITS on WWW client // Proc. of ICCE'98. — Beijing, AACE, 1998. — P. 324–327.
109. Jameson A. Numerical uncertainty management in user and student modeling: An overview of systems and issues // User Modeling and User Adapted Interaction. — 1996. — Vol. 5, N 3-4. — P. 193–251.
110. Jameson A. What can the rest of us learn from research on adaptive hypermedia — and vice versa? Tech. rep., University of Saarbrucken, 1999.
111. Joerding T. A temporary user modeling approach for adaptive shopping on the Web // Proc. of the 2nd Workshop on Adaptive Systems and User Modeling on the World Wide Web. — Eindhoven University of Technology, 1999. — P. 75–79.
112. Jones K.S., Willet P., Eds. Readings in Information Retrieval. — Morgan Kaufmann, 1997.
113. Kasyanov V. SVM — Siberian virtual museum of informatics history // Innovation and the Knowledge Economy: Issues, Applications, Case Studies. — Amsterdam: IOS Press, 2005. — Part 2. — P. 1014–1021.
114. Kasyanov V.N., Kasyanova E.V. Web-based systems for supporting computer-science teaching and learning // ACM SIGCSE Bulletin. — New York: ACM Press, 2002. — Vol. 34, N 3. — P. 238. — (Proc. of the 7th ACM SIGCSE Conf. on Innovation and Technology in Computer Science Education)
115. Kasyanov V.N., Kasyanova E.V. An environment for Web-based education of programming // HCI International 2003. — Heraklion: Crete University Press, 2003. — P.179–180. — (Proc. of the 10th Internat. Conf. on Human-Computer Interaction).

116. Kasyanova E.V. WAPE: an adaptive environment for Web-based education of programming // Proc. of the 17th IMACS World Congress — Paris, 2005. — P. 681–686.
117. Kay J., Kummerfeld B. User models for customized hypertext // Lect. Notes Comput. Sci. — 1997. — Vol. 1326. — P. 47–69.
118. Kay J., Kummerfeld R. An individualised course for the C programming language // Proc. of the 2nd Internat. World Wide Web Conf. — Chicago, 1994.
119. Kinshuk, Han B., Hong H., Ratel A. Student adaptivity in TILE // IEEE Intern. Conf. on Advanced Learning Technologies. — IEEE Computer Society, 2001. — P. 297–300.
120. Kobsa A. User modeling in dialog systems: Potentials and hazards. // AI & Society. — 1990. — Vol. 4, N 3. — P. 214–240.
121. Kobsa A. User modeling: Recent work, prospects and hazards. // Adaptive User Interfaces: Principles and Practice. — Amsterdam: North Holland Elsevier, 1993.
122. Kobsa A., Pohl W. The user modeling shell system BGP-MS // User Modeling and User-Adapted Interaction — 1995. — Vol. 4, N 2. — P. 59–106.
123. López J.M., Millán E., Pérez-de-la-Cruz J.L., Triguero F. ILESA: a Web-based intelligent learning environment for the simplex algorithm // Proc. of CALISCE'98, The 4th Intern. Conf. on Computer Aided Learning and Instruction in Science and Engineering. — Göteborg, 1998. — P. 399–406.
124. Martin J., Van Lehn, J. OLAE: Progress toward a multi-activity, Bayesian student modeler // Proc. of Artificial Intelligence in Education AIED. — Edinburgh, 1993. — P. 441–417.
125. Mislevy R., Gitomer D. The role of probability-based inference in an intelligent tutoring system // User Modeling and User-Adapted Interaction. — 1996. — Vol. 5. — P. 253–282.
126. Nelson T. Replacing the printed word: A complete literary system // Proc. IFIP Congress. — Netherlands, 1980. — P. 1013–1023.
127. Nielsen J. Multimedia, Hypertext und Internet: Grundlagen und Praxis des elektronischen Publizierens. — Vieweg, 1995.
128. Nissen H., Jeusfeld M., Jarke M., Zemanek G., and Huber, H. Requirements analysis from multiple perspectives: Experiences with conceptual modeling technology // IEEE Software. — 1996. — Vol. 13, N 2.
129. Nyce J., and Kahn P. A Machine for the Mind: Vannevar Bush's Memex. // From Memex to Hypertext: Vannevar Bush and the Mind's Machine. — Academic Press, 1991.
130. Okazaki Y., Watanabe K., Kondo H. An implementation of an intelligent tutoring system (ITS) on the World-Wide Web (WWW) // Educational Technology Research. — 1996. — Vol. 19, N 1. — P. 35–44.
131. Paiva A., Machado I. Vincent, an autonomous pedagogical agent for on-the-job training // Lect. Notes in Comput. Sci. —1998. — Vol. 1452. — P. 584–593.
132. Pearl J. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. — Morgen Kaufmann Publishers, 1988.

133. Pérez T., Gutiérrez J., Lopistéguy P. An adaptive hypermedia system // Proc. of the 7th World Conf. on Artificial Intelligence in Education (AI-ED'95). — Washington: DC, 1995. — P. 351–358.
134. Prentzas J., Hatzilygeroudis I., Koutsojannis C. A Web-based ITS controlled by a hybrid expert system // IEEE Intern. Conf. on Advanced Learning Technologies. — IEEE Computer Society, 2001. — P. 131–134.
135. Rich E. User modeling via stereotypes // Cognitive Science. — 1978. —N 3. — P. 329–354.
136. Ritter S. PAT-Online: A model-tracing tutor on the World-Wide Web // Proc. of Workshop "Intelligent Educational Systems on the World Wide Web" at AI-ED'97, The 8th World Conf. on Artificial Intelligence in Education — Kobe, ISIR, 1997. — P. 11–17.
137. Rosis F.D., Pizzutilo S., Russo A., Berry D.C., Molina F. J. Modeling the user knowledge by belief networks // User Modeling and User Adapted Interaction. — 1992. — Vol. 2. — P. 367 — 388.
138. Russell S., Norvig P. Artificial Intelligence: a Modern Approach. — Prentice Hall, Englewood Cliffs, NJ, 1995.
139. Schafer R., Weyrath T. Assessing temporally variable user properties with dynamic Bayesian networks // Proc. of the 6th Internat. Conf. on User Modeling (UM97). — Sardinia, 1997.
140. Schank R., Cleary C. Engines for Education. — Lawrence Erlbaum Associates, 1994.
141. Specht M. Empirical evaluation of adaptive annotation in hypermedia // ED-Media and ED-Telekom. — Freiburg, 1998.
142. Specht M., Oppermann R. ACE — adaptive courseware environment // The New Review of Hypermedia and Multimedia. — 1998. — Vol. 4. — P. 141–161.
143. Taylor J.C. Fifth generations distance education // The 20th ICDE World Conf. on Open learning and Distance Education. — Düsseldorf, 2001.
144. Vassileva J. A task-centered approach for user modeling in a hypermedia office documentation system // User Modeling and User-Adapted Interaction. — 1996. — Vol. 6, N 2–3.
145. Virvou M., Tsiriga V. Web-passive voice tutor: an intelligent computer assisted language learning system over the WWW // IEEE Intern. Conf. on Advanced Learning Technologies. — IEEE Computer Society, 2001. — P. 131–134.
146. Wang T., Hornung C. The Modular Training System (MTS) — a system architecture for Internet-based learning and training. — Fraunhofer: Fr-IGD, 1997.
147. Warendorf K., Tan C. ADIS — An animated data structure intelligent tutoring system or Putting an interactive tutor on the WWW // Proc. of Workshop "Intelligent Educational Systems on the World Wide Web" at AI-ED'97, The 8th World Conf. on Artificial Intelligence in Education. — Kobe: ISIR, 1997. — P. 54–60.
148. Weber G. Episodic learner modeling // Cognitive Science. —1996. — Vol. 20. — P. 195–236.

149. Weber G., Mollenberg A. ELM programming environment: A tutoring system for lisp beginners // Cognition and Computer Programming. — Ablex Publishing Corporation, 1995. — P. 373–408.
150. Weber G., Specht M. User modeling and adaptive navigation support in WWW-based tutoring systems // Proc. of the 6th Internat. Conf. on User Modeling (UM97). — Sardinia, 1997. — P. 289–300.
151. Wirth N. From Modula to Oberon // Software — Practice and Experience. — 1988. — Vol. 18, N 6. — P. 661–670.
152. Wirth N. The programming language Oberon // Software — Practice and Experience. — 1988. — Vol. 18, N 6. — P. 671–690.
153. Wu H., De Bra P., Aerts A., Houben G.-J. Adaptation control in adaptive hypermedia systems // Lect. Notes Comput. Sci. — 2000. — Vol.1892. — P. 250–259.
154. Wu H., De Kort E., De Bra P. Design issues for general-purpose adaptive hypermedia systems. // Proc. of the ACM Conf. on Hypertext and Hypermedia. — Aarhus, 2001. — P. 141–150.
155. Wu H., Houben G.J., De Bra P. Supporting user adaptation in adaptive hypermedia applications. // On-line Conf. and Informatiewetenschap 2000. — De Doelen, Rotterdam, 2000.
156. Zoonon programming language & compiler. Официальный сайт разработчиков языка. — http://www.zonnon.ethz.ch.

Приложение

ЯЗЫК ПРОГРАММИРОВАНИЯ ZONNON

Zonnon является новым языком программирования в семействе языков Паскаль, Модула-2 и Оберон, работа над которым ведется в Цюрихском институте информатики [101, 102]. Данное приложение содержит полное описание языка Zonnon в его текущей версии. Компилятор и среду программирования для данной версии языка можно скачать с сайта разработчиков языка Zonnon [156]. Там же всегда можно найти последние новости о языке, его реализации и применении, а также обратиться за консультацией к разработчикам.

1. Введение

Унификация абстракций является стержнем проектирования языка Zonnon. Она отражается в его четырех столпах: *модуль (module)* — текстовый контейнер, а также объект композиции программ; *объект (object)* — типовой образец для определяемых объектов; *определение (definition)* — концепция абстракции и композиции для определяемых интерфейсов; *реализация (implementation)* — контейнер для переиспользуемых фрагментов объектных реализаций. Указанные единицы языка обеспечивают базис для композиции программ при программировании в большом, а также для текстуального разбиения и раздельной компиляции во время разработки программы — они являются «гражданами первого сорта» в языке.

Объектная модель в языке Zonnon основывается на концепции, что «все есть объект». Она поддерживает три точки зрения на объекты: во-первых, как сущности с некоторым внутренним типом, использующие абстрактные операции способом безопасным для типов, во-вторых, как провайдеры служб, доступные через определенные интерфейсы и, в-третьих, как автоматические агенты, взаимодействующие через формальные диалоги. *Активности (Activities)* используются как для добавления поведения объектам, так и для реализации диалога. Они внедряют естественным образом параллелизм в язык.

Многие из концепций языка Zonnon получены им по наследству. Цель состояла в том, чтобы предложить выразительные и связующие черты, которые уже доказали свою ценность. Язык Zonnon также вводит некоторые новые черты, такие как перегрузка знаков операций для естественного представления математических и других выражений и обработка ситуаций

для повышения надежности. Некоторые черты были реанимированы им из ранних членов семейства паскалеподобных языков, например, пары *definition*, *implementation* и типы перечисления — из языка Модулы-2 и, по прагматическим соображениям, основная форма операторов вызова процедур *read* и *write* — из языка Паскаля.

Когда осуществляется выбор языка для построения современных программных систем, важным соображением является достижение взаимодействия между программами, написанными на разных языках. Язык Zonnon специально проектируется как платформно-независимый язык, поддерживающий такое взаимодействие.

2. Конструирование программ

Zonnon-программы основываются на четырех конструкциях: модуль, объект, определение и реализация.

Модуль (*module*) имеет двойственную природу: он объявляет синтаксический контейнер для логически связанных программных деклараций, и одновременно он определяет объект, чей жизненный цикл управляется системой. Таким образом, модуль предоставляет механизм для текстуального разбиения исходной программы, а также динамической загрузки на этапе выполнения некоторой части программы в виде созданного экземпляра объекта.

Любое число динамически созданных объектов может иметь свои жизненные циклы, управляемые системой, однако в любое заданное время только единственный экземпляр каждого модульного объекта может быть создан системой. Поскольку модуль формирует единицу инкапсуляции и скрытия данных, он идеален как контейнер для реализации абстрактных типов данных.

Объектом (*object*) является типовой образец, включающий в себя поля, методы и активности. Поля представляют состояние объекта, методы – его функциональность, а активности — его параллельное поведение. Он может выставлять свой интерфейс к системному окружению двумя способами. Во-первых, с помощью присущего ему интерфейса (*intrinsic interface*), т.е. множества всех элементов, которые программист выбрал в качестве публичных, а не приватных, и, во-вторых, большим числом определений (*definitions*), каждое из которых выставляет свой аспект (*facet*), представляющий собой некоторый взгляд на службы объекта со стороны клиентов.

Определение (*definition*) задает некоторый свой аспект (*facet*) объекта в терминах абстрактного интерфейса, включающего определения полей и

сигнатуры методов. Определения могут образовывать не только иерархию, но и сеть связанных типов (*a network of related types)*.

Реализация (*implementation*) определяет агрегат реализаций полей и методов, предназначенных для переиспользования, при присоединении к программе с помощью одного или нескольких объектных образцов. Объект, реализующий определение, должен реализовывать все его поля и методы. Однако если объект импортирует реализацию определения с тем же именем, то это неявно предполагает, что она будет его (возможно частичной) реализацией.

Программный текст (*program text*) состоит из модулей, объектов, определений и реализаций. Внутренний интерфейс (*intrinsic interface*) программы представляет собой множество деклараций, сделанных публичными всеми ее частями. Программа периода исполнения (*runtime program*) состоит из одного или более модулей и любых объектов, которые создаются динамически. Система (*system*) предоставляет механизмы динамической программной загрузки и выгрузки модулей и динамическим управлением объектных ресурсов во время выполнения, когда происходит исполнение программы.

Эти конструкции используются для формирования всей структуры программы в виде программных *единиц module, object*, *definition* и *implementation*. Каждая конструкция может существовать как отдельно скомпилированная единица (*separately compiled unit*) или может текстуально встраиваться в содержимое другой конструкции. Между этими конструкциями выполняется ряд отношений, которые определяют, как они могут совместно использоваться; они представляют собой следующие типы отношений, где каждый из *x* и *y* изображает некоторую конструкцию:

- *x содержит (contains) y.* Конструкция *x* может содержать одну или более конструкций *y*, текстуально в нее вложенных;
- *x импортирует (imports) y.* Конструкция *x* может импортировать декларации из одной или более конструкций *y*;
- *x агрегирует (aggregates from) y.* Конструкция *x* может импортировать фрагменты реализации из конструкции *y*;
- *x реализует (implements) y.* Если идентичны имена *definition* и *implementation,* то *implementation* предоставляет, по крайней мере, реализацию *definition*, иначе она может предоставлять реализацию для одной или больше *definition,*
- *x уточняет (refines) y. Definition x* уточняет *definition y,* опуская, добавляя или модифицируя службы.

Правила для допустимого использования конструкций (программных единиц) проиллюстрированы на рис. 1, они имеют следующий вид:

- единица *module* может иметь конструкции *definition*, *implementation* и *object,* текстуально в нее вложенными;
- единицы *module*, *definition*, *implementation* и *object* могут импортировать декларации из других единиц *module*, *definition* и *object;*
- единицы *module*, *implementation* и *object* могут агрегировать из других единиц *implementation;*
- единицы *module*, *implementation* и *object* могут реализовывать конструкции *definition;*
- конструкции *definition* могут уточнять другие конструкции *definition.*

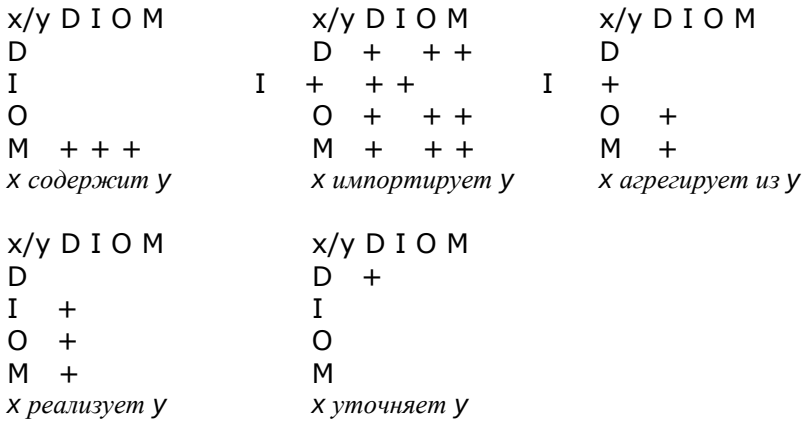


Рис. 1. Представление допустимых отношений между конструкциями (программными единицами) в виде матриц, в которых используются следующие обозначения: *D = definition, I = implementation, O = object, M = module,* + — отношение допустимо.

3. Синтаксическая нотация

Синтаксис языка Zonnon определяется с использованием Расширенного Бекуса-Наура Формализма (Extended Backus-Naur Formalism, EBNF).

3.1. Определение Расширенного Бекуса-Наура Формализма

Нотация Расширенного Бекуса-Наура Формализма (РБНФ), используемая в данном описании языка, характеризуется следующими свойствами:

- альтернативы разделяются символом,
- скобки [и] обозначают факультативность выражения в скобках,
- скобки { и } обозначают повторение содержимого (возможно 0 раз),
- скобки (и) используются для формирования групп элементов,
- нетерминальные символы начинаются с прописной буквы (например, *Statement*),
- терминальные символы либо начинаются со строчной буквы (например, *letter)*, либо записывается целиком полужирным шрифтом (например, **begin**), или представляются в виде строк (например, ":="),
- комментарии начинаются с символа // и продолжаются до конца строки.

3.2. Описание РБНФ с помощью РБНФ

В качестве примера можно определить синтаксис РБНФ с помощью самого РБНФ следующим образом:

```
Syntax = {Production}.
Production = NonTerminalSymbol "=" Expression ".".
Expression = Term {"|" Term}.
Term = Factor {Factor}.
Factor = terminalSymbol | NonTerminalSymbol |
      "(" Expression ")" | "[" Expression "]" | "{" Expression "}" .
```

3.3. Описание РБНФ

Конструкции РБНФ описаны ниже:

3.3.1. Последовательность

```
A = BC.
```

A состоит из *B,* за которым следует *C*

Примеры:

```
Sentence = Subject Predicate.
FileName = Name '.' Extension.
Name = FirstName Surname.
```

3.3.2. Повторение

```
A = {B}.
```

A состоит из нуля или более символов *B*.

Примеры:

```
File = {Record}.
Bill = {Item Price}.
```

3.3.3. Выбор

```
A = B | C.
```

A состоит из *B* или *C*.

Примеры:

```
Fork = Resource | Data.
Meal = Breakfast | Lunch | Dinner.
```

3.3.4. Факультативность

```
A = [B].
```

A состоит из *B* или пусто.

Пример:

```
SelectedDrink = [ Tea | Coffee | Chocolate ]. // Возможно отсутствие!
```

3.3.5. Кавычки и полужирный шрифт

Текст в кавычках и текст, выделенный полужирным шрифтом, изображает сам себя.

Примеры:

```
ImportDeclaration = import Import {"," Import}.
OwnSymbol = "me" | self.
```

4. Языковые символы и идентификаторы

4.1. Словарь и представление

Лексемами языка Zonnon являются идентификаторы, числа, строки, знаки операций и разделители. Имеются следующие лексические правила:

- пробелы и символы конца строки не должны встречаться внутри лексем и игнорируются, хотя и являются существенными для разделения двух соседних лексем (за исключением их использования в

комментариях и в строках);

- строчные и прописные символы рассматриваются как различные.

4.2. Идентификаторы

Идентификаторы представляют собой последовательности букв, цифр и подчерков "_". Первый символ идентификатора должен быть буквой или подчерком.

```
ident = ( letter | "_" ) { letter | digit | "_" }.
```

Примеры:

```
X Scan Zonnon GetSymbol firstLetter
_external_package27 // подчерк обычно используется для взаимодействия
с другими языками
```

4.3. Модификаторы и спецификаторы

Модификатор (*modifier*) используется для указания альтернативной семантики, когда один и тот же синтаксис используется для более чем одной цели. Он представляет собой список слов, чисел и символов, заключенный в скобки { }.

Примеры:

```
{ value }
{ public }
```

Спецификатор (*specifier*) используется для предоставления дополнительной информации, такой как тип ожидаемого объекта или его размер. Он представляет собой список чисел, заключенный в скобки { }, либо спецификацию протокола в РБНФ нотации (См. также 10.3).

Примеры:

```
var r: real{32};
i := integer(t);
{ bodypart = LEG | NECK| ARM.}
```

4.4. Численные константы

Числами являются (беззнаковые) целые и вещественные константы. Если константа специфицирована с суффиксом *H*, представление является шестнадцатеричным, иначе представление является десятичным. Вещественное число всегда содержит десятичную точку и факультативно может содержать десятичный масштабный множитель. Буква *E* означает «возвести

десять в степень». Числовая константа факультативно может завершаться модификатором ширины, имеющим вид (заключенного в скобки) числа битов, которые должны использоваться для ее представления. Если ширина не специфицирована, используется ее значение по умолчанию, определенное в деталях реализации компилятора. Для более полной информации о типах см. 13.1.

```
number = (integer | real) [ "{" Width "}" ].
integer = digit {digit} | digit {hexDigit} "H".
real = digit {digit} "." {digit} [ScaleFactor].
ScaleFactor = "E" ["+" | "-"] digit {digit}.
hexDigit = digit | "A" | "B" | "C" | "D" | "E" | "F".
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
Width = ConstExpression.
```

Целая константа совместима как с целыми (знаковыми) типами, так и с кардинальными (беззнаковыми) типами.

Примеры:

Константа	*Тип*	*Значение*
1991	integer/cardinal	1991
0DH{8}	integer{8} cardinal{8}	13
12.3	real	12.3
4.567E8	real	456700000
0.57712566E-6{64}	real{64}	0.00000057712566

4.5. Символьные константы

Символьная константа — это литера, заключенная в одинарные (') или двойные (") кавычки. Открывающая кавычка должна совпадать с закрывающей и не должна быть сама литерой. Символьная константа может также обозначаться порядковым кодом литеры в шестнадцатеричной нотации, за которым следует литера X. Это является полезным для представления специальных литер, которые либо не могут быть напечатаны, либо принадлежат расширенному множеству литер.

```
CharConstant = "'" character "'" | '"' character '"' | digit { HexDigit } "X".
character = letter | digit | Other.
Other = // Любая литера из алфавита, отличная от тех, которые в использовании…
```

Примеры:
"a" 'n' """ '"' 20X

4.6. Строковые константы

Строковые константы представляют собой неизменные последовательности литер, заключенные в одинарные (') или двойные (") кавычки. Открывающая кавычка должна совпадать с закрывающей и не должна появляться в строке. Количество символов в строке называется ее длиной. Строка из одного символа (длины 1) может использоваться везде, где разрешено использовать символьную константу, и наоборот. Строковая константа может присваиваться переменной типа *string* (см. 6.3.1).

```
string = '"' { character } '"' | """ { character } """.
character = letter | digit | Other.
Other = // Любой символ алфавита, кроме тех, которые используются в качестве ограничителей
```

Примеры:
"Zonnon" "Don't worry!" "x" 'hello world'

4.7. Зарезервированные слова, ограничители и знаки операций

Знаки операций и разделители — это специальные литеры, пары литер или зарезервированные слова, перечисленные ниже.

4.7.1. Зарезервированные слова

Следующие зарезервированные слова (напечатанные полужирным шрифтом) не могут использоваться в качестве идентификаторов и состоят исключительно из строчных букв:

accept activity array as await begin by case const definition div do else elsif end exception exit false for if implementation implements import in is launch loop mod module new nil object of on operator or procedure receive record refines repeat return self send then to true type until var while

или целиком из прописных букв:

ACCEPT ACTIVITY ARRAY AS AWAIT BEGIN BY CASE CONST DEFINITION DIV DO ELSE ELSIF END EXCEPTION EXIT FALSE FOR IF IMPLEMENTATION IMPLEMENTS IMPORT IN IS LAUNCH LOOP MOD MODULE NEW NIL OBJECT OF ON OPERATOR OR PROCEDURE RECEIVE RECORD REFINES REPEAT RETURN SELF SEND THEN TO TRUE TYPE UNTIL VAR WHILE

4.7.2. Ограничители

Литерами-ограничителями являются:
( ) [ ] { } . (точка) , (запятая) ; (точка с запятой) : (двоеточие) .. (диапазон)
| (разделитель вариантов) ' (одинарная кавычка) " (двойная кавычка)

4.7.3. Предопределенные знаки операций

Предопределенными знаками операций являются:

- (унарный минус) + (унарный плюс) ~ (отрицание)
^ (унарное разыменование)
+ - * / **div mod** & **or**
:= (присваивание) = (равно) # (не равно) < <= > >= **in implements**

4.7.4. Определенные пользователем знаки операций

Язык Zonnon вводит понятие определенных пользователем знаков операций. Они объявляются подобно процедурам (См. 6.3).

4.8. Комментарии

Комментарии можно вставить между любыми двумя символами в программе. Они имеют вид произвольных последовательностей литер, заключенных в скобки (* и *). Комментарии могут вкладываться одни в другие. Они не влияют на смысл программы.

5. Описания

5.1. Описание идентификаторов и правила области действия

Каждый идентификатор, встречающийся в программе, должен быть введен в объявлении, если он не является предопределенным. Объявление специфицирует также определенные перманентные свойства элемента: является ли он константой, типом, переменной (см. 5.4) или процедурой (см. 8). Идентификатор затем используется в качестве ссылки на соответствующий элемент.

Область действия идентификатора простирается текстуально от точки его объявления до конца блока, к которому принадлежит данное объявление и, следовательно, по отношению к которому оно локально. Она исключает области действия идентификаторов с одинаковыми именами, которые описаны во вложенных блоках. Правила областей действия имеют вид:

- ни один идентификатор не может обозначать более одного элемен-

та в некоторой заданной области действия (т.е. ни один идентификатор не может быть объявлен дважды в блоке);
- на идентификатор можно ссылаться только внутри его области действия;
- идентификаторы, являющиеся именами полей объекта или его методов/процедур допустимы только в обозначениях (designators) объекта, где они должны квалифицироваться именем объекта.

```
QualIdent = { ident "." } ident.
```

Примеры:

```
Month.Oct (* см 5.3.2 *)
NameSpace.Program
```

5.1.1. Модификаторы объявлений

Объявление может иметь факультативный модификатор. Модификаторы объявлений определяются следующим образом:

- *private*: идентификатор видим только внутри области действия его объявления;
- *public*: идентификатор видим внутри области, в которой он объявлен, и в любой конструкции, которая явно импортирует программную конструкцию, которая содержит это объявление;
- *immutable*: используется для переменной вместе с *public* и указывает на то, что значение переменной только читается вне области действия, в которой она объявлена.

Пример:

```
var {private} flag: boolean;
var {public, immutable} refCount: integer; (*доступ только для чтения*)
```

5.2. Объявление констант

Объявление константы связывает идентификатор с константным значением.

```
ConstantDeclaration = ident "=" ConstExpression.
ConstExpression = Expression.
```

Примеры:

```
const N = 10;
      limit = 2*N – 1; (* см. 6.2.2*)
      fullSet = {min(set) .. max(set)}; (* см. 5.3.1)
```

Константное выражение — это выражение, которое может быть вычислено при простом текстовом просмотре, без реального выполнения программы. Его операндами должны быть константы или вызовы предопределенных функций.

5.3. Объявление типа

Тип данных определяет множество значений, которые допустимы у переменных данного типа, и операций, которые к ним применимы. Объявление типа связывает идентификатор с типом. В случае структурных типов (массивов и объектов) оно также определяет структуру переменных этого типа. Типы объектов определяются в 5.3.4 и 11.1

```
TypeDeclaration = ident "=" Type.
Type = ( TypeName [ Width ] | EnumType | ArrayType | ProcedureType | InterfaceType ).
Width = "{" ConstExpression "}".
```

5.3.1. Базисные типы

Базисные типы обозначаются предопределенными идентификаторами. Ассоциированные знаки операций определены в 6.2, предопределенные функции и процедуры в 9.

Значения базисных типов следующие:

- object — родовой тип, из которого выведены объектные типы,
- boolean — логические значения *true* и *false,*
- char — символы лежащего в основе реализации литерного множества,
- cardinal — положительные целые числа из интервала от *min*(*cardinal*) до *max*(*cardinal*),
- integer — целые числа из интервала от *min*(*integer*) до *max*(*integer*),
- fixed — большие числа с фиксированной точностью между *min*(*fixed*) и *max*(*fixed*),
- real — вещественные числа между *min*(*real*) и *max*(*real*),
- set — множества чисел (целых или кардинальных) между 0 и *max*(*set*),
- string — литерные строки.

Заметим, что *object* является зарезервированным словом. Для типов *char*, *integer*, *real* и *set* количество битов, необходимых для хранения значе-

ния, может быть специфицировано указанием целого числа битов в виде константного значения в скобках { } после имени типа — так называемой ширины (width) типа. По умолчанию используются следующие значения ширины для базисных типов: char{16}, cardinal {32}, integer{32}, real {80}, set{32}, fixed {128}. О приведении между различными типами см. 6.3.8.

5.3.2. Перечисляемые типы

Перечисление является типом, который состоит из именованного списка идентификаторов, обозначающих значения, которые составляют тип. Эти идентификаторы квалифицируются именем типа при использовании в качестве именованных констант в программе. Значения упорядочены, и их отношение порядка определяется их текстуальной последовательностью в перечислении. Нет других значений, принадлежащих данному типу. Порядковый номер первого значения равен нулю и увеличивается на единицу при переходе от любого элемента перечисления к следующему идентификатору.

```
EnumType = "(" IdentList ")".
IdentList = ident { "," ident }.
```

Примеры:

```
type NumberKind = (Bin, Oct, Dec, Hex);
     Month = (Jan, Feb, Mar, Apr, May, Jun, July, Sep, Oct, Nov, Dec);
```

Имена в различных перечислениях не обязаны различаться, поскольку всегда квалифицируются при использовании. Для рассмотренного примера *NumberKind.Oct* всегда отличается от *Month.Oct*.

Значения выражений могут конвертироваться к некоторому другому типу (см. 5.3.9).

Предопределенная функция *pred* возвращает значение предшественника значения перечисления, указанного в качестве ее параметра, для всех значений, за исключением первого значения перечисления. Предопределенная функция *succ* возвращает значение преемника значения перечисления, указанного в качестве ее параметра, для всех значений, отличных от последнего значения перечисления.

5.3.3. Типы массивов

Массив является структурой, состоящей из некоторого числа элементов, которые все имеют один и тот же тип, называемый типом элементов. Массивы могут индексироваться либо положительным целым числом, либо значением перечисляемого типа. В первом случае количество элементов в объявлении массива определяет его длину. Элементы массива обозначаются индексами, которые представляют собой целые числа между нулем и длиной массива минус единица. Во втором случае имя перечисляемого типа используется в объявлении массива, а элементы массива обозначаются значениями перечисляемого типа.

Синтаксические правила для типа массивов имеют вид:

```
ArrayType = array Length {"," Length} of Type.
Length = Expression | "*".
```

Массивы могут быть многомерными; это означает, что элементы массива сами могут быть массивами, и в принципе, допустимо соединение форм спецификации различных длин. Но эта возможность вполне может быть ограничена реализацией. Примером и контрпримером являются:

```
type Acceptable = array * of array 42 of T; (* array *, 42 of T *)
     Jagged = array 42 of array * of T; (* 'зубчатый' массив *)
```

Описание **array** m **of array** n **of** T текстуально эквивалентно **array** m, n **of** T.

Например, **array** * **of array** 42 **of** T может быть записано **array** *, 42 **of** T

Выражение *len(a, n)* возвращает количество элементов по измерению *n* массива *a*. Выражение *len(a)* является сокращением для *len(a, 0)*.

В массиве количество элементов по любому измерению может быть переменным, и тогда оно обозначается звездочкой. Ответственность за отведение области памяти в куче с использованием зарезервированного слова *new* для каждого экземпляра массива ложится на программиста:

```
arrayVariable := new ArrayType(length0, length1, … );
```

Значение длины должно выражаться положительным выражением целого типа, и количество таких выражений должно соответствовать числу из-

мерений данной переменной.

Примеры использования массивов:

```
type Vector = array * of integer;
procedure CreateAndReadVector(var a: Vector);
      var i, n: integer;
begin
      read(n);
      a := new Vector(n);
      for i := 0 to len(a) – 1 do
                read(a[i])
      end
end CreateAndReadVector;
procedure InitializeMatrix(var mat: array *, * of real);
      var i, j: integer;
begin
      for i := 0 to len(mat, 0) - 1 do
                for j := 0 to len(mat, 1) – 1 do
                          mat[i, j] := 0.0
                end
          end
end InitializeMatrix;
…
var m: array 10, 10 of real;
…
InitializeMatrix(m);
```

5.3.4. Строковый тип

Переменная типа *string* представляет неизменные последовательности литер. Строки можно сравнивать на равенство и неравенство с использованием операций ‘=’ и ‘#’. Знак операции ‘+’ представляет конкатенацию, а ‘:=’ — присваивание. Предопределенная процедура *copy* конвертирует значения строкового типа в значения литерного массива и обратно.

5.3.5 Объектные типы

Объект *(object)* является шаблоном типа данных, состоящим из полей, методов и активностей. Поля представляют состояние объекта, методы — его функциональность, а активности — его параллельные активности. Он может раскрывать свой интерфейс системному окружению двумя способами. Во-первых, посредством интерфейса его внутреннего типа (на который ссылаются, как на его внутренний интерфейс), состоящего из множества всех элементов, которые программист решил сделать общими, а не приват-

ными, и, во-вторых, некоторым числом определений, каждое из которых раскрывает некоторый свой аспект служб объекта по отношению к его клиентам.

```
Object = object [ ObjModifier ] ObjectName [ FormalParameters ] [ ImplementationClause ] ";"
        [ ImportDeclaration ]
        Declarations
        { ActivityDeclaration }
        ( BlockStatement | end ) SimpleName.
ObjModifier = "{" ident "}". // value или ref
        // private или public
ActivityDeclaration = activity ActivityName [ ImplementationClause ] ";"
        Declarations ( BlockStatement | end SimpleName ).
ImplementationClause = implements DefinitionName { "," DefinitionName }.
ImportDeclaration = import Import { "," Import } ";".
Import = ImportedName [ as Ident ].
ImportedName = ( ModuleName | ImplementationName | NamespaceName |
        DefinitionName, ObjectName).
```

Объект образован объявлениями, включающими константы, типы, переменные (на которые ссылаются как на поля), и процедуры (на которые ссылаются как на методы). Модификаторы *public* и *private* можно использовать для объявления видимости содержимого объекта. Если модификатор не представлен, то умолчанию считается *private*.

Отдельные данные можно сделать общими явным использованием модификатора {*public*}, размещенным после их объявления. Сам объект тоже может иметь модификатор, который обозначает его либо как объект-значение, либо ссылочный объект, используя значения модификаторов *value* и *ref* соответственно. Если модификатор отсутствует, по умолчанию выбирается *value*.

Переменные, которые являются ссылочными объектами, представляют собой ссылки на объекты, которые создаются динамически во время выполнения программы с использованием *new*. Объект факультативно может иметь параметры, которые используются в теле объекта для инициализации полей, когда создается экземпляр с использованием *new*.

```
object {ref} Box(w, h: integer);
      var width, height: integer;
      procedure Area(): integer;
      begin
            return width * height
      end Area;
begin
      self.width := w; self.height := h (* self является здесь необязательным в обоих случаях*)
end Box.
...
```

```
var box: Box;
...
box := new Box(3, 7); (*создает новый объект Box с шириной 3 и высотой 7 *)
```

См. 11.2 об *объектах* как о программных единицах.

5.3.6. Типы записей

Запись является типом объектов значений. Она может использоваться для инкапсуляции описаний констант, типов и переменных, но не методов и активностей. Ключевое слово **record** эквивалентно **object** {value}. Переменные, которые объявлены как объекты-значения, статически размещаются во время компиляции

Примеры:

```
record Position; (*описание Position типа 'record'*)
      var x, y: integer
end Position;
```

что эквивалентно

```
object {value} Position; (*описание Position типа 'record'*)
      var x, y: integer
end Position;
```

```
record Date; (*описание Date типа 'record' *)
      var year: integer{8};
      month: Month;
      day: integer{8}
end Date;
```

5.3.7. Типы постулированных интерфейсов

Интерфейс является постулированной реализацией для объекта, образованного из одного или более определений. См. 5.3.8 и 11.2 для дальнейших деталей.

```
InterfaceType = object [ PostulatedInterface ].
PostulatedInterface = "{" DefinitionName { "," DefinitionName } "}".
```

5.3.8. Процедурные типы

Переменная процедурного типа *T* имеет некоторую процедуру или метод *P* или *nil* в качестве своего значения. Если процедура *P* присвоена переменной типа *T*, списки формальных параметров у *P* и *T* должны соответствовать друг другу по множеству правил, описанных в 13.4. *P* не может быть предопределенной процедурой или локальной по отношению к другой процедуре.

Когда метод присваивается переменной типа процедуры, он должен иметь префиксом обозначение экземпляра объекта, его содержащего.

```
ProcedureType = procedure [ ProcedureTypeFormals ].
ProcedureTypeFormals = "(" [ PTFSection { ";" PTFSection } ] ")" [ ":" FormalType ].
PTFSection = [ var ] FormalType { "," FormalType }.
FormalType = { array "*" of } ( TypeName | InterfaceType ).
InterfaceType = object [ PostulatedInterface ].
PostulatedInterface = "{" DefinitionName { "," DefinitionName } "}".
```

5.3.9. Преобразование типов

В языке Zonnon преобразование (conversion) типа внутри одного ‘семейства’ (такого как *integer*) является неявным тогда, когда оно гарантированно является безопасным. Однако любое преобразование типов между семействами должно быть явным (поскольку оно вызывает преобразование внутреннего представления). Обратные преобразования (например, *integer*{32} в *integer*{16}) внутри семейства также всегда должны быть явными. Механизм исключительных ситуаций обнаруживает конвертационные аномалии (см. 8.10.1). Взаимосвязь между типами базируется на модели Общей Системы Типов ECMA, как она используется в .NET, и суммирована в следующей таблице:

Семейство типов	*Размер в байтах*								
	8		16		32		64		128
fixed									M
								↗	
real					M	→	M		
				↗		↗1			
integer	M	→	M	→	M	→	M		
		↗		↗		↗			
cardinal	M	→	M	→	M	→	M		
			↑	↗					
char			M						

M – *уполномоченный тип для соответствующей реализации,*

→ – *всегда допустимое неявное преобразование типа (внутри одного семейства),*

↗, ↑ – *всегда допустимое явное преобразование (изменяет представление),*

1 – *может привести к снижению точности представления.*

Заметим, что отношение неявного преобразования типов является транзитивным и обратное отношение (противоположное направлению стрелки) требует явного преобразования и может привести к усечению или исключительной ситуации.

5.3.9.1. Имя типа, используемое в качестве функции преобразования. Чтобы осуществить преобразование типа, имя целевого типа может рассматриваться как встроенная функция, которая берет значение выражения исходного типа в качестве параметра и возвращает конвертированное значение. Факультативный второй параметр этой функции указывает размер результата.

Синтаксис:

```
TypeConversion = TypeIdentifier "(" expression [ "," Width ] ")".
```

Примеры:

```
integer(x + e/f, 16)
```

является значением выражения *x + e/f* , представленным в виде 16-битового целого (может возникнуть исключительная ситуация, если преобразование типа не возможно).

```
integer(x + e/f)
```

является значением выражения *x + e/f* , представленным в виде 32-битового целого (предполагая, что 32 является размером по умолчанию для целого типа). Заметим, что целые не могут неявно быть приведены к вещественным и поэтому:

```
var count, sum: integer; mean: real;
...
mean := sum / count
```

синтаксически *не* допустимо и требует явного преобразования типа:

```
mean := real(sum) / real(count)
```

5.3.9.2. Неявный тип константы. Тип простой целочисленной константы определяется путем описания переменной, которой она присваивается. Так, например, при наличии описания:

```
var i: integer {16};
```

оператор присваивания

```
i := 1;
```

фактически рассматривается компилятором как присваивание

```
i := 1{16};
```

Если размер не специфицирован, то выбирается тот размер, который определен для данного типа по умолчанию.

Другое преобразование типа достигается посредством предопределенных процедур (см. раздел 9).

5.4. Описание переменных

Переменная обладает некоторым значением, которое она может получить от выражения с помощью операции присваивания (см. 7.1). Переменная определяется как имеющая некоторый тип, который не может быть изменен; тип определяет то множество значений, которыми переменная может обладать. Описание переменной вводит переменные путем определения для каждой из них идентификатора и типа данных.

```
VariableDeclaration = IdentList ":" Type.
```

Примеры:

```
var i, j, k: integer;
  x, y: real;
  p, q: boolean;
  s: set {32};
  a: array 100 of real;
  name: array 32 of char;
  size, count: integer;
  mousePosition: Position;
  dateOfBirth, today: Date;
```

6. Выражения

Выражением является конструкция, которая специфицирует вычисление. В выражении константы и текущие значения переменных комбинируются для вычисления других значений с помощью применения операций и процедур функций. Выражение состоит из операндов и знаков операций; можно использовать скобки для того, чтобы выразить специфические ассоциации знаков операций и операндов. Типы промежуточных значений, используемых во время вычисления выражения, являются прерогативой реализации. Тип результата выражения определен в разделе, посвященном совместимости выражений (см. 13.6).

6.1. Операнды и обозначения

За исключением конструкторов множеств и литеральных констант (числа, символьные константы или строки), операнды изображаются обозначениями. Обозначение состоит из идентификатора, ссылающегося на константу, переменную или процедуру. Возможно, что этот идентификатор квалифицируется идентификатором, обозначающим модуль, определение,

реализацию или объект и может завершаться селекторами, если обозначаемый объект является элементом структуры.

```
Designator = Instance
      | Designator "{" Type "}"// Приведение
          | Designator "^" // Разыменование
          | Designator "[" Expression { "," Expression } "]"// Элемент массива
          | Designator "(" [ ActualParameters ] ")" // Вызов функции
          | Designator "." MemberName // Селектор члена
Instance = (self| InstanceName | DefinitionName "(" InstanceName ")" ).
ActualParameters = Actual { "," Actual }.
Actual = Expression [ "{" [var] FormalType "}" ]. // Аргумент с сигнатурой типа
```

Символ ^ используется для того, чтобы факультативно сделать явной ссылку в программном тексте.

Примеры:

Обозначение	*Тип*	*Значение*
Size	integer	Значение переменной, называемой *size*
a[i]	real	Элемент массива *a* в позиции *i*
dateOfBirth.day	integer{8}	Поле *day* объекта, называемого *dateOfBirth*
w[3].name[i]	char	Элемент в позиции *i* в поле *name* у элемента массива *w* в позиции 3

Если *a* обозначает массив, то *a*[*e*] обозначает тот элемент *a,* чей индекс является текущим значением выражения *e*. Выражение *e* должно быть либо типом перечисления, либо кардинального или целого типа. Обозначение вида *a*[*e*0*, e*1*, …., en*] означает *a*[*e*0][*e*1]….[*en*].

Если *obj* обозначает объект, то *obj.f* обозначает поле *f* у *obj* или метод *f* у объекта *obj* (см. 11.1).

Если обозначаемый объект является константой или переменной, то обозначение ссылается на ее текущее значение. Если он является процедурой, обозначение ссылается на эту процедуру. Однако если объект является процедурой-функцией и за ним следует (возможно, пустой) список параметров, это обозначение приводит к активации этой процедуры и представляет значение, являющееся результатом этого исполнения. Фактические параметры должны соответствовать формальным параметрам, как в подходящих вызовах процедуры (см. 8.2).

6.2. Предопределенные знаки операций

Предопределенные знаки операций зафиксированы и встроены в язык.

6.2.1. Логические операции

Эти знаки операций применимы к операндам *boolean* и производят результат *boolean*.

or — логическая дизъюнкция	p **or** q — "если *p* то *true*, иначе *q*"
& — логическая конъюнкция	p & q — "если *p* то *q*, иначе *false*"
~ — отрицание	~ p — "не *p*"

6.2.2 Арифметические операции

Знаки операций +, -, и * применимы в выражениях к операндам числовых типов (см. 6.3.1). Знак операции деления применим к операндам типа *real* и производит результат типа *real*. При использовании знаков унарных операций, - обозначает изменение знака и + обозначает тождественную операцию.

+ —	сложение
- —	вычитание
* —	умножение
/ —	вещественное деление (вещественных)

Примеры:

```
i :=j + k;
x := real(i) / float(j); (* см. раздел 5.5.9*)
```

Знаки операций *div* и *mod* применимы только к целым и кардинальным операндам.

div —	целочисленное деление
mod —	модуль

Они удовлетворяют следующим формулам, определенным для любого *x* и положительного делителя *y*:

```
x = (x div y) * y + (x mod y)
0 <= (x mod y) < y
```

Если значение делителя отрицательно, то смысл знаков операций *div* и *mod* определяется реализацией. Программисту рекомендуется проверять это условие и использовать математические рассуждения для обеспечения гарантии того, что используются в качестве делителей только положитель-

ные значения.

Примеры:

x	*y*	*x* **div** *y*	*x* **mod** *y*
5	3	1	2
-5	3	2	1

6.2.3. Знаки операций для множеств

Знаки операций для множеств применимы к операндам типа *set* и вырабатывают результат типа *set*. Объявленные размеры в битах для операндов типа *set* должны совпадать. Знак унарного минуса обозначает дополнение *x*, т.е. *–x* обозначает множество целых чисел между 0 и *max(set)*, которые не являются элементами *x*.

\+ — объединение (покомпонентное или),
\- — разность (покомпонентное вычитание),
($x - y = x * (-y)$)
* — пересечение (покомпонентное и),
/ — симметричная разность множеств (покомпонентное исключенное или).
($x / y = (x-y) + (y-x)$)

Конструктор множества определяет значение множества перечислением в скобках его элементов, если они есть. Элементы должны быть числами из диапазона 0 .. *max*(*set*). Диапазон *m* .. *n* обозначает все целые числа из интервала, начинающегося элементом *m* и завершающегося элементом *n*, включая *m* и *n*. Если *m* > *n*, то *m* .. *n* обозначает пустое множество.

Примеры использования множеств:

```
const left = 0; right = 1; top = 2; bottom = 3;
var edges: set; x, y: integer;
begin
      edges := { }; (* пустое множество *)
      if x < xMin then edges := edges + {left}
      …
      if left in edges then … (* отрезать слева *)

const opCodemask = {0..3};
var opCode, word: set;
…
      opCode := word * opCodeMask; (* извлечь op-code *)
```

6.2.4. Отношения

Отношения вырабатывают результат *boolean*. Отношения =, #, <, <=, >, и >= применимы числовым типам и *char*. Отношения = и # применимы к *boolean* и *set*, так же как и к процедурным типам (включая значение **nil**). *x* **in** *s* означает "*x* является элементом *s*". *x* должно быть целого типа, а *s* - типа *set*.

= — равно
— неравно
< — меньше
<= — меньше или равно
> — больше
>= — больше или равно
in — быть элементом множества
is — *x* **is** *T* является **true**, если присущий тип *x* является *T*
implements — *x* **implements** *D* равно **true**, если объект *x* реализует определение *D*

Примеры выражений

Выражение	*Тип*	*Пояснения*
1991	integer	Простое константное значение
i **div** 3	integer	Целочисленное деление *i* на 3
~wellFormed or outOfRange	boolean	(не well-formed) или out-of-range
(i+j) * (i-j)	integer	Арифметическое выражение
s - {8, 9, 13}	set{8}	*s* , из которого удалены 8, 9, 13
keys **in** {left, right}	boolean	*keys* в *left* или в *right* или в том и другом
('0'<=ch) & (ch<='9')	boolean	*ch* является цифрой

6.3. Определенные пользователем знаки операций и объявления знаков операций

Перегрузка операций вводит понятие определяемых пользователем операций и позволяет использовать обычный синтаксис в выражениях, их содержащих. Знаки операций определяются только в некотором модуле, реализующем абстрактный тип, т.е. таком, который определяет некоторый новый определяемый пользователем тип и множество операций на нем. Типично это используется, когда вводятся новые типы данных, такие как комплексные числа или матрицы.

6.3.1. Базисные знаки операций, для которых допустима перегрузка

Множество предопределенных операций, для которых допустимы перегрузки, является следующим:

```
- (унарный минус) + (унарный плюс) ~
^ (унарное разыменование)
+ - * / div mod & or
= # < <= > >= in
:= (присваивание является специальным случаем, см. 6.3.3)
```

Заметим, что для знаков операций *implements* и *is* не допустима перегрузка (См. 11.1).

6.3.2. Объявления новых знаков операций

Перегруженные (полиморфные) операции вводятся как объявления знаков операций. Синтаксис объявления имеет следующий вид:

```
OperatorDeclaration = operator [ OpModifiers ] OpSymbol [ FormalParameters ] ";"
      OperatorBody ";".
OperatorBody = Declarations BlockStatement OpSymbol.
OpModifiers = "{" ident { "," ident } [ "," Priority ] "}".
Priority = ConstExpression.
OpSymbol = String. // 1- или 2-литерная строка; множество возможных символов
      ограничено
```

Пример:

```
operator '+' (x1, x2: Complex): Complex;
      var res: Complex;
begin
      res.re := x1.re + x2.re;
      res.im := x1.im + x2.im;
      return res
end '+';
```

Для перегруженных знаков операций количество параметров в объявлении знака операции должно быть тем же, что и для предопределенной операции с тем же знаком.

В определенном пользователем знаке операции для присваивания должно быть два параметра, первый из которых должен подставляться по ссылке.

Имеется возможность определять перегруженные операции только в *module*, но не в *object* или *definition*. Причина — должна быть полная возможность статического разрешения перегрузки во время компиляции. Другая причина — ясно разделить две концепции: объекты, реализующие интерфейсы (определения) и абстрактные типы данных с ассоциированными операциями.

Объявления знаков операций можно сделать видимыми извне модуля, где они объявлены. В этом случае законно использовать указанные знаки операций в нормальных выражениях наряду с предопределенными знаками операций в единицах, импортирующих данный модуль. Компилятор ответственнен за выбор правильной версии знака операций в каждом случае.

Допускается определять знаки операций в некотором модуле для расширения абстрактного типа данных. Эти знаки операций должны быть определены в терминах операций, уже определенных в модуле, где объявлен абстрактный тип данных.

Обычно все импортируемые вхождения должны квалифицироваться именами импортируемых единиц. Это возможно, но не требуется для знаков операций. Например, имеется два законных способа использования “новых сложений” для операндов некоторого типа *T*:

```
module M;
      type T {public}= ...;
      operator {public}"+" ( a, b : T ) : T; begin ...end "+";
end M.

object Obj;
      import M;
      var x, y : T;
begin
      x := x + y; (* подобно обычному выражению *)
      x := x M."+" y; (* полностью квалифицированный *)
end Obj.
```

Процедура знака операции не может быть вызвана как обычная функция:

```
x := M."+"(x, y); (* не законно; должна использоваться нотация выражения *)
```

6.3.3. Правила, управляющие перегрузкой

Следующее множество правил применяется для перегруженных знаков операций.

1. По крайней мере, тип у одного операнда перегруженного знака операции должен быть определенным пользователем типом (тип массивов, тип объектов, процедурный тип, тип перечисления). Законно вводить определенные пользователем версии “базисных” типов, такие как *integer*, *real* и *boolean*.

2. Не разрешена спецификация типа объекта с постулированным интерфейсом (таким как *object {D}*) в качестве параметра знака операции. Причина — должна быть полная возможность разрешения перегрузки во время компиляции (т.е. статически).

3. Нет ограничений на тип результата перегруженной операции.

4. Число аргументов, старшинство перегруженной операции и форма (префиксная или постфиксная) унарной операции должны совпадать с соответствующими чертами для предопределенной операции с тем же знаком.

5. Конструкция разыменования со знаком ^ (см. обозначение правила Designator в синтаксисе) рассматривается здесь как постфиксный знак унарной операции. Следовательно, любая перегрузка знака операции ^ сохраняет форму унарного постфиксного знака; аналогично, унарные + и – знаки операций всегда являются унарными префиксными знаками операций.

6. Допускается перегрузка оператора присваивания. В этом случае присваивание рассматривается как специальная операция со знаком ‘:=’, реализующая определенный побочный эффект и не производящая значения.

7. В перегруженном знаке операции для оператора присваивания должны быть два параметра, и первый параметр должен подставляться по ссылке.

8. Законно специфицировать более одной версии перегруженной операции с одним и тем же знаком; в этом случае типы параметров соответствующих определений операции должны отличаться от любого другого определения операции с тем же знаком (см. 6.3.1).

6.4. Старшинство операций

При использовании в выражениях знаков операций синтаксически выделяются четыре их класса, различающихся по старшинству (силе связывания). Операции одного и того же старшинства соединяются слева направо. Например, *x-y-z* означает (*x-y*)-*z*. Старшинство операций от наибольшего старшинства к наименьшему имеет следующий вид:

1) унарная операция отрицания ~ ,
2) операции типа умножения,
3) операции типа сложения,
4) отношения.

Знаки операций используются в выражениях:

```
Expression = SimpleExpression
        [ ( "=" | "#" | "<" | "<=" | ">" | ">=" | in ) SimpleExpression ]
        | Designator implements DefinitionName.
SimpleExpression = [ "+"|"-" ] Term { ( "+" | "-" | or ) Term }.
Term = Factor { ( "*" | "/" | div | mod | "&" ) Factor }.
Factor = number
        | CharConstant
        | string
        | nil
        | Set
        | Designator
        | new TypeName [ "(" ActualParameters ")" ]
        | new ActivityInstanceName
        | "(" Expression ")"
        | "~" Factor.
Set = "{" [ SetElement { "," SetElement } ] "}".
SetElement = Expression [ ".." Expression ].
```

Допустимые знаки операций перечислены ниже. Некоторые из них применимы к операндам различных типов, обозначая различные операции. В этих случаях реальная операция идентифицируется типом операндов. Операнды должны быть выражениями, совместимыми по отношению к данному знаку операции (см. 13.6).

6.5. Численные разрешения внутри выражений

Выражение состоит из серии выполнений операций над своими операндами. Для каждой операции отношение между разрешением каждого из его операндов и результатом определяется следующим образом:

Операция	Первый операнд	Второй операнд	Результат
+	integer{s}	integer{t}	integer{max(s, t)}
-	integer{s}	integer{t}	integer{max(s, t)}
*	integer{s}	integer{t}	integer{s + t}
div	integer{s}	integer{t}	integer{s}
mod	integer{s}	integer{t}	integer{t}

+	cardinal{s}	cardinal{t}	cardinal{max(s, t)}
-	cardinal{s}	cardinal{t}	cardinal{max(s, t)}
*	cardinal{s}	cardinal{t}	cardinal{s + t}
div	cardinal{s}	cardinal{t}	cardinal{s}
mod	cardinal{s}	cardinal{t}	cardinal{t}
+	real{s}	real{t}	real{max(s, t)}
-	real{s}	real{t}	real{max(s, t)}
*	real{s}	real{t}	real{s + t}
/	real{s}	real{t}	real{ s + t}
+	fixed	fixed	fixed
-	fixed	fixed	fixed
*	fixed	fixed	fixed
/	fixed	fixed	fixed

где max(s, t) = s, if s > t else t

7. Операторы

Операторы обозначают действия. Имеются элементарные и структурные операторы. Элементарные операторы не составляются из таких частей, которые сами являются операторами. Они являются присваиванием, вызовом процедуры, операторами *await*, *return* и *exit*. Структурные операторы образуются из частей, которые сами являются операторами. Они используются для выражения последовательного, условного, селективного и повторного выполнения. Оператор может быть также пустым, в этом случае он обозначает отсутствие действия. Пустой оператор включается для того, чтобы ослабить правила пунктуации в последовательности операторов.

```
Statement = [ Assignment
          | ProcedureCall
          | IfStatement
          | CaseStatement
          | WhileStatement
          | RepeatStatement
          | LoopStatement
          | ForStatement
          | await Expression
          | exit
          | return [ Expression ]
          | BlockStatement
          | launch Statement
          | Send
          | Receive ].
```

Последовательность операторов (statement sequence) обозначает последовательность действий, специфицированных компонентными операторами, которые разделяются точкой с запятой.

```
StatementSequence = Statement {";" Statement}.
```

Пример:

```
temp := a; a := b; b := temp; (*обмен значениями между a и b*)
```

7.1. Оператор присваивания

Оператор присваивание (assignment statement) заменяет текущее значение переменной некоторым новым значением, специфицированным выражением. Выражение должно быть совместимым с переменной по присваиванию (см. 13.4). Знак операции присваивания записывается как “:=” и произносится “становится”.

```
Assignment = Designator ":=" Expression.
```

Примеры:

```
i := 0;
p := i = j;
x := i + 1;
k := log2(i+j);
F := log2;
s := {2, 3, 5, 7, 11, 13};
a[i] := (x+y) * (x-y);
t.key := I;
w[i+1].name := "John";
t := c;
```

7.2. Вызовы процедур

В *module* вызов процедуры активизирует процедуру. Когда она объявлена в *object,* на процедуру ссылаются как на метод. В каждом из этих случаев он может содержать список фактических параметров, которые заменяют соответствующие формальные параметры, определенные в объявлении процедуры (см. 8). Соответствие устанавливается позиционно – относительным порядком параметров в списках фактических и формальных параметров. Имеется два вида параметров: параметры-переменные и параметры-значения.

Если формальный параметр является параметром-переменной, соответствующий фактический параметр должен быть обозначением, изображающим переменную. Если он изображает элемент структурной переменной, селекторы компонент вычисляются, когда имеет место подстановка формальных/фактических параметров, т.е. до исполнения процедуры. Если формальный параметр является параметром-значением, соответствующий фактический параметр должен быть выражением. Это выражение вычисляется до активации процедуры, и результирующее значение присваивается формальному параметру.

```
ProcedureCall = Designator.
```

Примеры:

```
WriteInt(i*2+1)
INC(w[k].count)
t.Insert("John")
```

Вызов метода состоит из имени объекта, за которым следует точка и затем имя некоторой процедуры, описанной в объявлении типа объекта для данного объекта. В методе зарезервированное слово *self* ссылается на объект, в котором вызывается метод.

Вызов конкретной процедуры может также быть “предохранен (safeguarded)” путем прибавления в качестве префикса объекта в определении. Например:

```
object T implements I, D; … end T;
var t: T;
```

Клиент, который хочет иметь специфическое использование интерпретации *t* службы, специфицированной *D* (т.е. как супервызов (*supercall)*), мог бы тогда просто вызвать методы и поля *D*, предохраненные с помощью *t*:

```
D(t).f(..); ..; .. := D(t).x;
```

Порядок, в котором вычисляются параметры во время вызова процедуры/метода, определен в деталях реализации компилятора (*Compiler Implementation Details)*.

7.3. Оператор if

```
IfStatement =
    if Expression then StatementSequence
        {elsif Expression then StatementSequence}
        [else StatementSequence]
        end.
```

Пример:

```
if (ch >= "A") & (ch <= "Z") then ReadIdentifier
elsif (ch >= "0") & (ch <= "9") then ReadNumber
elsif (ch = " ' ") or (ch = ' " ') then ReadString
else SpecialCharacter
end
```

Оператор *if* специфицирует условное выполнение охраняемой последовательности операторов. Выражение, предшествующее операторной последовательности, называется ее предохранителем, и его тип должен быть *boolean*. Предохранители вычисляются в порядке их вхождения, если один из них принимает значение *true*, выполняется ассоциированная с ним последовательность операторов. Если не удовлетворяется ни один из предохранителей, выполняется последовательность операторов, следующая за символом *else*, если он есть.

7.4. Оператор case

Оператор *case* специфицирует выбор и исполнение последовательности операторов в соответствии со значением выражения. Вначале вычисляется выражение *case*; затем вычисляется последовательность операторов, чья *case*-метка содержит полученное значение. Выражение *case* должно быть целого или кардинального типа, который совместим (см. 13.6) с типами всех меток *case*, либо как выражение *case*, так и метки *case* должны иметь тип *char* или перечисления. Метки *case* являются константами, и ни одно значение метки не должно встречаться более одного раза. Если значение выражения не встречается в качестве ни одной метки *case*, то либо выбирается последовательность операторов, следующая за символом *else*, если он есть, либо возникает исключительная ситуация *UnmatchedCase*.

```
CaseStatement = case Expression of Case {"|" Case} [else StatementSequence] end.
Case = [CaseLabelList ":" StatementSequence].
CaseLabelList = CaseLabels {"," CaseLabels}.
CaseLabels = ConstExpression [".." ConstExpression].
```

Пример:

```
case ch of
        "A" .. "Z": ReadIdentifier (*предполагается последовательное кодирование букв*)
      | "0" .. "9": ReadNumber
      | "'", '"': ReadString
    else SpecialCharacter
end

case month of
        Month.Apr, Month.Jun, Month.Sep, Month.Nov: days := 30
      | Month.Feb:    if Leap(year)
                      then days := 29
                      else days := 28
                      end
        else days := 31
end
```

7.5. Оператор while

Оператор *while* специфицирует повторяющееся выполнение последовательности операторов, пока выражение типа *boolean* (ее предохранитель) сохраняет *true*. Предохранитель проверяется перед каждым выполнением последовательности операторов, и таким образом, последовательность операторов будет выполнена нуль или более раз.

```
WhileStatement = while Expression do StatementSequence end.
```

Примеры:

```
var i, k, idNumber: integer;
...
while i # 3 do writeln('Hello'); i := i + 1 end

read(idNumber);
while ~Valid(idNumber) do
        write('Type ID number again ');
        read(idNumber)
end;
(* Valid(idNumber) *)
while i > 0 do i := i div 2; k := k + 1 end

while (t # nil) & (t.key # i) do t := t.left end
```

7.6. Оператор repeat

Оператор *repeat* специфицирует повторяющееся выполнение последовательности операторов до тех пор, пока не выполнится условие, специфицированное выражением типа *boolean*. Указанная последовательность операторов выполняется, по крайней мере, один раз.

RepeatStatement = **repeat** StatementSequence **until** Expression.

Примеры:

```
var idNumber: integer;
repeat
        write ('Type ID number '); read(idNumber)
until Valid(idNumber);
...
var i, x: integer; buffer: array 10 of integer;
...
i := 0;
(* конвертировать неотрицательное значение x в десятичное представление *)
repeat buffer[i] := x mod 10; x := x div 10; inc(i) until x = 0;
(* распечатать цифры в правильном порядке *)
repeat dec(i); write(char(buffer[i] + integer("0"))) until i = 0
```

7.7. Оператор for

Оператор *for* специфицирует повторяющееся исполнение последовательности операторов фиксированное число раз, пока присваиваются последовательные значения переменной типа *integer*, называемой управляющей переменной оператора *for*.

ForStatement = **for** ident ":=" Expression **to** Expression [**by** ConstExpression] **do** StatementSequence **end**.

Пример:

```
var i : integer;
...
for i := 0 to 79 do k := k + a[i] end
for i := 79 to 1 by -1 do a[i] := a[i-1] end
```

Оператор

```
for v := low to high by step do statements end
```

эквивалентен

```
v := low; temp := high;
if step > 0 then
      while v <= temp do statements; v := v + step end
else
      while v >= temp do statements; v := v + step end
end
```

Значение выражения *low* должно быть совместимым по присваиванию с переменной *v*, и *high* должно быть выражением, совместимым с *v*. Значение *step* (*шаг*) должно быть ненулевым константным выражением целого типа. Если опущен *шаг*, тогда *шаг* по умолчанию равен 1.

7.8. Оператор loop

Оператор *loop* специфицирует повторяющееся выполнение последовательности операторов. Он завершается выполнением оператора *exit* в этой последовательности.

```
LoopStatement = loop StatementSequence end.
```

Пример:

```
loop (* копировать целые числа из input в output до первого отрицательного*)
      read(i);
      if i < 0 then exit end;
      write(i)
end
```

Оператор *loop* полезен для описания повторений с несколькими точками выхода или в тех случаях, когда условие выхода естественно разместить в середину последовательности операторов.

Оператор *exit* обозначается символом *exit*. Он специфицирует завершение объемлющего оператор *loop* и продолжение исполнения с оператора, следующего за этим оператором *loop*. Оператор *exit* контекстно, а не синтаксически ассоциируется с оператором *loop,* который его содержит.

7.9. Оператор return

Оператор *return* указывает на завершение процедуры. Он обозначается символом *return*, за которым следует выражение, если процедура является процедурой-функцией. Тип выражения должен быть совместим по присваиванию (см. 13.4) с типом результата, специфицированным в процедуре.

Процедура-функция требует наличия оператора *return*, указывающего результирующее значение. В собственно процедурах оператор *return* реализуется концом тела процедуры. Следовательно, явный оператор *return* появляется как дополнительная (возможно исключительная) точка завершения.

7.10. Операторы блока и запуска

Оператор блока позволяет группировать логически связанные операторы и вводить обработчики исключительных ситуаций. Блоки могут быть вложенными:

```
BlockStatement = begin [ BlockModifiers ]
                 StatementSequence
                 { ExceptionHandler }
                 [ CommonExceptionHandler ]
                 end.
BlockModifiers = "{" ident { "," ident } "}". // locked, concurrent
ExceptionHandler = on ExceptionName { "," ExceptionName } do StatementSequence.
CommonExceptionHandler = on exception do StatementSequence.
```

Исполняется последовательность операторов в блоке.

7.10.1. Обработка исключительных ситуаций

Если возникла исключительная ситуация, тогда проверяются обработчики в их текстуальном порядке до тех пор, пока не встретится обработчик этой ситуации или не будет достигнута общая исключительная ситуация. Тогда выполняется последовательность операторов, соответствующая найденному обработчику.

Имена исключительных ситуаций имеют вид предопределенных идентификаторов и содержат:

- ZeroDivision: деление на нуль;
- Overflow: значение не лежит в диапазоне min(*type*)..max(*type*);
- OutOfRange: индекс массива выходит за границы;
- NilReference: неинициализированный экземпляр массива/объекта/канал активности;
- UnmatchedCase: поток управления достиг отсутствующего ***else*** в операторе ***case***;
- Conversion: некорректное приведение типа (не гарантируется "t ***is*** тип");
- Read: ошибочно сформатированное входное значение для *read* или *readln.*

Пример:

```
var idNumber: integer; idValid: boolean;
begin
        read(idNumber);
        if Valid(idNumber) then idValid := true; Process(idNumber)
        else idValid := false (*неправильное число*)
        end
on exception do
        idValid := false (*неправильный сорт литер*)
end
```

7.10.2. Модификаторы параллельности и операторы запуска

Факультативно блок может иметь модификатор. Определены следующие модификаторы:

- *locked*: только одна активность разрешена внутри области действия блока. Он используется для осуществления взаимного исключения для защищенного доступа к переменным в параллельных программах. Операторы внутри блока выполняются последовательно.
- *concurrent*: отдельные операторы в блоке могут выполняться параллельно на одном или нескольких процессорах в любом порядке. Однако если некоторый оператор имеет префиксом слово **launch**, то он становится оператором запуска (launch). Это предоставляет способ определения порядка, в котором параллельные операторы начинают свое исполнение. Блок завершается одновременно с полным завершением выполнения последнего оператора.

В обоих случаях ограничители *begin* и *end* используются в качестве барьера.

Пример:

```
begin {locked}
... (*операторы в блоке выполняются последовательно, но отдельными единицами *)
end
```

```
begin {concurrent}...launch S; T; launch U; ... end
```

Действие этого оператора состоит в запуске *S*, затем исполнении *T*, затем запуске *U* и ожидании в **end,** когда завершатся все запущенные операторы. Это обеспечивает некоторую новую параллельность операторного уровня, которая позволяет программистам специфицировать ‘логику запусков’ без требования, чтобы операторы выполнялись последовательно.

7.11. Оператор await

Оператор *await* используется для условного планирования внутри активности в объекте или модуле.

await Expression

Он должен встречаться внутри оператора блока с модификатором *locked*. Выражение определяет предусловие продолжения выполнения.

Когда он выполняется, вычисляется логическое выражение, и если результат есть *true,* то выполнение продолжается со следующего оператора. Однако, если результат есть *false*, то выполнение приостанавливается до тех пор, пока системный планировщик последовательно не перевычислит условие (возможно неоднократно) и найдет, что оно стало равным *true*. Тогда выполнение продолжится со следующего оператора.

Пример: буфер *FIFO.*

Данный пример показывает, как буфер *FIFO* можно реализовывать с использованием объекта. Предполагается, что производитель, который ‘помещает’ данные, принадлежит другой активности, чем потребитель, который ‘забирает’ их. Операторы *await* регулируют содержимое буфера. Модификаторы *locked* предохраняют от совместного доступа к разделяемым буферам, когда они изменяются, чтобы сохранить их целостности.

```
object Buffer;
(*Первым-пришел первым-ушел буфер ('магазин') *)
        const bufLen = 1000;
        var data: array bufLen of integer;
              in, out: integer;
        procedure {public} Put (i: integer); (*поместить элемент в буфер *)
        begin {locked}
                await (in + 1) mod bufLen # out; (*ждать пока не полон *)
                data[in] := i;
                in := (in + 1) mod bufLen
        end Put;
        procedure {public} Get (var i: integer); (*взять элемент из буфера *)
        begin {locked}
                await in # out; (* ждать пока не пуст *)
                i := data[out];
                out := (out + 1) mod bufLen
        end Get;
        begin
              in := 0; out := 0;
end Buffer.
```

7.12. Оператор send

Оператор *send* может использоваться при реализации активности (см. раздел 10) для направления значения в диалог, установленный между двумя активностями. Оператор не является блокирующим, т.е. исполнение оператора, следующего за ним, продолжается непосредственно сразу после начала его исполнения.

Send = **send** expression ["=>" activity].

Примеры:

```
send pi*x/180.0 => a
(*Конвертировать градусы в радианы и послать результат вызываемой активности 'a' *)
send "29 August 2003" (* Послать строку с датой назад вызвавшей активности*)
```

7.13. Оператор receive

Оператор *receive* может использоваться внутри реализации активности (см. раздел 10) для получения значения из диалога. Исполнение блокируется до тех пор, пока не станет значения, доступного для получения.

Receive = **receive** [activity "=>"] variable.

Примеры:

```
receive a => date     (* Получить строку date из вызываемой активности 'a' *)
receive angle     (* Получить значение angle из вызвавшей активности 'a' *)
```

7.14. Оператор accept

Оператор *accept* используется в реализации активности (см. раздел 10) для принятия значения из диалога. Оператор *accept* является не блокирующим, он возвращает значение, как только оно становится доступным, или иначе возвращает *nil*. В любом случае исполнение непосредственно продолжается с оператора, который следует за оператором *accept*.

Accept = **accept** [activity "=>"] variable.

Примеры:

```
accept a => date (*Получить строку date из вызываемой активности 'a', или nil, если
сейчас ничего сейчас не доступно*)
accept angle (*Получить значение angle из вызвавшей активности 'a'*)
```

8. Объявления процедур (и методов) и формальные параметры

Объявление процедуры состоит из заголовка процедуры и тела процедуры. Заголовок специфицирует идентификатор процедуры и формальные параметры, если они есть. Тело содержит объявления и операторы. Идентификатор процедуры повторяется в конце объявления процедуры. Процедура, объявленная в объекте, называется методом.

Имеется два вида процедур: собственно процедуры и процедуры функции. Последние активируются обозначениями функции как частей выражений и вырабатывают результаты, которые являются операндами этих выражений. Собственно процедуры активируются вызовами процедур. Процедура является процедурой функцией, если ее формальные параметры специфицируют тип результата. Тело процедуры функции должно содержать оператор *return*, который определяет этот результат.

Все константы, переменные, типы и процедуры, объявленные в теле процедуры являются локальными по отношению к процедуре. Поскольку процедуры могут быть также объявлены как локальные элементы, процедурные объявления могут быть вложенными (предмет для ограничений при реализации). Вызов процедуры в ее объявлении приводит к рекурсивной активации.

В дополнение к ее формальным параметрам и локально описанным элементам, элементы, описанные в окружении процедуры, также видны в процедуре (за исключением тех элементов, которые имеют те же имена, что и элементы, объявленные локально).

```
ProcedureDeclaration = ProcedureHeading [ ImplementationClause ] ";" [ ProcedureBody ";" ].
ProcedureHeading = procedure [ ProcModifiers ] ProcedureName [ FormalParameters ].
ProcModifiers = "{" ident { "," ident } "}". // private, public, sealed
ProcedureBody = Declarations BlockStatement SimpleName.
FormalParameters = "(" [ FPSection { ";" FPSection } ] ")" [ ":" FormalType ].
FPSection = [ var ] ident { "," ident } ":" FormalType.
```

Формальные параметры являются идентификаторами, объявленными в списке формальных параметров процедуры. Они соответствуют фактическим параметрам, специфицированным в вызове процедуры. Это соответствие устанавливается, когда происходит вызов процедуры. Имеются два вида параметров: параметры-значения и параметры-переменные, указанные в списке формальных параметров с помощью отсутствия или наличия ключевого слова ***var***. Параметры-значения являются локальными переменными, которым присваиваются в качестве начальных значения соответствующих фактических параметров. Параметры-переменные соответствуют фактиче-

ским параметрам, являющимся переменными, и они означают эти переменные. Область действия формального параметра простирается от его объявления до конца блока процедуры, в которой он объявлен. Процедура функция без параметров должна иметь пустой список параметров. Она должна вызываться обозначением функции, список фактических параметров которого также пуст.

Правила соответствия между формальными и фактическими параметрами следующие. Пусть *Tf* является типом формального параметра *f* (неоткрытый массив) и *Ta* — тип соответствующего фактического параметра *a*. Для параметра переменной тип *Ta* должен быть тем же, что и *Tf*, или *Tf* должен быть типом *object,* а *Ta* являться расширением *Tf*. Для параметров значений формальный параметр *a* должен быть совместим по присваиванию с *f* (см. 13.4).

Если *Tf* является открытым массивом, тогда *a* должна быть массивом совместимым с *f* (см. 13.5). Длина *f* берётся из *a*.

8.1. Модификаторы процедуры

Факультативно после ключевого слова *procedure* можно разместить модификатор для обозначения природы процедуры. Определены следующие модификаторы:

- *private*: процедура видна только в области действия, где она объявлена; это свойство действует по умолчанию;
- *public*: процедура видна в области действия, в которой она объявлена и внутри любой конструкции, которая импортирует конструкцию, в которой она объявлена;
- *sealed*: процедура не может быть далее переопределена (замещена), противоположность состоянию быть *sealed* называется как быть *open*.

Примеры:

```
procedure ReadInt(var x: integer);
    var i: integer; ch: char;
begin
    i := 0; read(ch);
    while ("0" <= ch) & (ch <= "9") do
            i := 10*i + (integer(ch) - integer("0")); read(ch)
    end;
    x := i
end ReadInt;
```

```
procedure {private} WriteHex(x: integer);
(* предусловие: 0 <= x <100000H *)
      var i: integer; buf: array 5 of integer;
begin
      i := 0;
      repeat buf[i] := x mod 10H; x := x div 10H; inc(i) until x = 0;
      repeat dec(i);
                  if buf[i] < 10 then write(char(buf[i] + integer("0")))
                  else write(char(buf[i] – 10 + integer("A")))
                  end
      until i = 0
end WriteHex;

procedure log2(x: integer): integer;
(* предусловие: x > 0 *)
      var y: integer;
begin
      y := 0;
      while x > 1 do x := x div 2; inc(y) end;
      return y
end log2;
```

8.2. Свойства

Свойство (*property)* является переменной, для которой процедуры доступа предоставляются программистом и автоматически вызываются всякий раз, когда ее значение читается или записывается. Всякий раз, когда происходит доступ к значению переменной в выражении, вызывается функция, помеченная модификатором *get*; всякий раз, когда значение переменной устанавливается присваиванием, вызывается процедура, помеченная модификатором *set*. Переменная, для которой предоставлена только функция *get*, является 'только для чтения' (‘read only’). Переменная, для которой предоставлена только процедура set, является 'только для записи' (‘write only’).

```
definition D;
      var x: T;
end D.
object O implements D;
      procedure {get} Getx ( ): T implements D.x;
      (* 'getter': вызываемая автоматически, когда происходит доступ к x *)
      begin
                  ...
                  return (...)
      end Getx;
      procedure {set} Setx (expression: T ) implements D.x;
      (* 'setter': вызывается автоматически, когда x получает значение выра-
          жения*)
```

```
    begin
            ...
    end Setx;
end O.
```

9. Предопределенные процедуры

Следующая таблица перечисляет предопределенные процедуры. Некоторые из них являются родовыми процедурами, т.е. они применимы к операндам разного типа. В спецификации *v* обозначает переменную, *x* и *n* — выражения, а *T* — тип. Имена предопределенных процедур могут быть записаны прописными буквами.

Название	*Тип(ы) аргумента(ов)*	*Тип результата*	*Предназначение*
abs(x)	integer, cardinal or real	type of x	абсолютная величина *x*
assert(b)	b: boolean	none	если ~b завершить
assert(b, n)	b: boolean; n: integer or cardinal	none	если ~b завершить, сообщить n среде
box(v)	value object	type of v	ссылка на значение объекта v
cap(x)	x: char	char	соответствующая прописная буква предусловие : x является буквой
copy(x, v)	x: string; v:character array	none	v := x
copy(v, x)	x: string; v:character array	none	x := v
copyvalue(v)	v:ref object	value object	разыменование объекта
dec(v)	v: integer, cardinal or enumeration type	none	v:= v - 1
dec(v, n)	v: integer, cardinal or enumeration type n: integer or cardinal type	none	v := v - n
excl(v, x)	v: set; x: integer or cardinal type	none	v := v - {x}
halt(n)	n: integer or cardinal const	none	завершить исполнение программы
inc(v)	v: integer, cardinal or enumeration	none	v := v + 1
inc(v, n)	v: integer, cardinal or enumeration n: integer or cardinal type	none	v := v + n
incl(v, x)	v: set; x: integer or cardinal type	none	v := v + {x}
len(v, n)	v:array; n: integer or cardinal const	integer	длина v по измерению n (первое измерение = 0)
len(v)	v:array	integer	эквивалентно len(v, 0)
low(x)	x:char	char	соответствующая строчная буква предусловие: x является буквой
max(T)	integer	integer	максимальное значение типа integer{w}
max(T)	cardinal	cardinal	максимальное значение типа cardinal{w}
max(T)	enumeration	enumeration	максимальное значение перечисления
max(T)	char{w}	integer	максимальная character
max(T)	real{w}	real	максимальное значение типа real{w}
max(T)	set{w}	integer	максимальный элемент set{w}
min(T)	integer	integer	минимальное значение типа integer{w}

min(T)	enumeration	enumeration	минимальное значение перечисления
min(T)	char{w}	integer	минимальная литера
min(T)	real{w}	real	минимальное значение типа real{w}
min(T)	set{w}	integer	0
odd(x)	x: integer	boolean	x **mod** 2 = 1
pred(x)	x: integer	integer	x – 1, предусловие: x # min(integer)
pred(x)	x: enumeration	type of x	предыдущее значение перечисления, предусловие: x # min(enumeration)
pred(x)	x: char	char	предыдущая литера, предусловие: x # min(char)
size(T)	any type	integer	число байтов, требуемых для T
succ(x)	x: integer or cardinal	integer	x + 1, предусловие: x # max(integer)
succ(x)	x: enumeration	type of x	следующее значение перечисления, предусловие: x # max(enumeration)
succ(x)	x: char	char	следующая литера, предусловие: x # max(char)
unbox(v)	v: ref object	type of v	объект значение из ссылочного объекта v

В *assert(x, n)* и *halt(n)* интерпретация *n* является реализационно зависимой.

О предопределенных процедурах ввода-вывода см. раздел 10.5.

10. Активности, поведение и взаимодействие

Объявление активности подобно объявлению метода (процедуры), в котором опущен список параметров. Зарезервированное слово *activity* используется, чтобы отличить объявление активности от объявления метода. Активности также могут иметь модификаторы *private* и *public* для управления их видимостью. Когда активность объявлена, ее экземпляры могут создаваться в любом активном объекте или модуле.

Семантически разница между активностью и методом более существенна. Активности объявлены и затем воплощены (запущены), а не вызваны, и новая активность размножается с каждым запуском.

Операция *new* используется, чтобы создать каждый экземпляр активности.

10.1. Поведение

Активности предоставляют средства инкапсулированного поведения, добавленного к объекту или модулю (рассматриваемому как отдельный объект). Объект может содержать произвольное число активностей или не содержать их вовсе, и в этом случае он является пассивным объектом. Обычно поведенческие активности являются приватными по отношению к объекту (или модулю), который содержит их, и создаются и размножаются конструктором.

Пример:

```
object Cell (*конвейера*);
        type Job = …;
        var in, out, n: integer;
              buf: array N of Job;
        procedure Get (j: Job);
        begin …
        end Get;
        procedure { public } Put (j: Job);
        begin …
        end Put;
        activity Process;
        var … (*пространство состояний активности*)
        begin …
        end Process;
        var p: Process;
begin
        n := 0; in := 0; out := 0;
        p := new Process (*Создать активность в Cell *)
end Cell;
```

10.2. Взаимодействие

Формальная синтаксическая спецификация может ассоциироваться с активностями для объектного взаимодействия. Для того чтобы начать взаимодействие, вызывающий вначале создает активность в вызываемом объекте, который неявно открывает диалог с ним, который затем начинается между вызывающей и вызванной активностями по формальному синтаксису диалога вызванной активности. Весьма важно, что взаимодействие между вызывающей и вызванной активностью является асимметричным. Вызывающий знает вызываемого по его имени, тогда как вызываемый не осведомлен об имени вызывающего, а знает только об ассоциации между ними, являющейся данным диалогом.

Реальный обмен синтаксическими лексемами между вызывающим и вызываемым управляется с помощью операций *send* и *receive*, описанных в разделах 8.12 и 8.13, где *receive* имеет родовой аргумент объекта. Если необходимо, знак операции *is* может тогда использоваться для распознавания разных типов синтаксических лексем (см. 6.2.4).

10.3. Протокол РБНФ

Определение активности может включать диалог, который является формальной синтаксической спецификацией коммуникационного протокола в РБНФ. Он представляется в виде модификатора к типу перечисления,

который определяет алфавит терминальных лексем синтаксиса. Название активности и ее тип перечисления образуют сигнатуру активности. Заметим, что в спецификации протокола в виде РБНФ к коммуникации элемента от вызывающего к вызываемому приставляется спереди “?”.

Пример:

```
definition Fighter;
    activity (* Синтаксис протокола, в данном случае он рекурсивный *)
        { fight = { attack ( { defense attack } | RUNAWAY [ ?CHASE] | KO | fight ) }.
        attack = ATTACK strike.
        defense = DEFENSE strike.
        strike = bodypart [ strength ].
        bodypart = LEG | NECK | HEAD.
        strength = integer. }
        Karate = (RUNAWAY, CHASE, KO, ATTACK, DEFENSE, LEG, NECK, HEAD);
end Fighter.

object Opponent implements Fighter;
    activity Karate implements Fighter.Karate;
        var t: object;
        procedure fight;
        begin
              while t is ATTACK do
                    receive t;
                    while t is DEFENSE do receive t; strike
                                if t is ATTACK then strike else halt(protocolError) end
                    end;
                    if t is RUNAWAY then
                                if (*не истощен*) then send Karate.CHASE end;
                                return (* борьба завершается *)
                    elsif t is KO then return (* борьба завершается *)
                    elsif t is ATTACK then fight (* рекурсия, продолжается борьба *)
                    else halt(protocolError) (* ошибка протокола *)
                    end
              end
        end fight;
        procedure strike;
        begin (* заметим использование типовых проверок в качестве охраны *)
              if (t is LEG) or (t is NECK) or (t is HEAD)
              then
                   receive t; (*часть тела *)
                   if t is integer then  receive t end (*факультативный параметр силы *)
              end
        end strike;
        begin (* Karate*)
                                receive t;
                                fight
        end Karate;
end Opponent.
```

```
object Challenger;
    import Opponent, Fighter;
    var opp: Opponent; f: Fighter.Karate;
    opp := new Opponent; (* создать оппонента *) …
    f := new opp.Fighter.Karate; (* создать диалог *)
    send Fighter.Karate.ATTACK => f; … (* борьба соответствующая протоколу диалога *)
…
end Challenger.
```

10.4. Завершение

Объект может только тогда завершиться, когда на него больше нет ссылок и когда завершены все его активности. Активность завершается после выполнения оператора, непосредственно предшествующего *end* в ее теле процедуры.

10.5. Процедуры ввода и вывода

Язык включает встроенные возможности для текстового ввода и вывода. Концептуально чтение и запись соответствуют получению и посылке лексем *из* и *в* предопределенные активности *standard input* и *standard output* соответственно.

Для удобства предоставлены предопределенные процедуры для чтения и выдачи текста в Паскале-подобном стиле. Процедурами для ввода текста являются *read* и *readln,* а для вывода — *write* и *writeln*. Все вводимое и выводимое должно быть текстом, причем неявно предполагается, что он должен изображаться как строки литер, разделенные символами конца строки.

10.5.1. Параметры и специальный синтаксис

Процедуры используются с нестандартным синтаксисом для их списков параметров. Допускается переменное число параметров, которые могут быть разных типов. Параметры типа *char* не требуют преобразования типов данных, однако для других типов (таких как целые, вещественные и др.) передача данных включает неявное преобразование типов данных.

10.5.2. Процедуры ввода

10.5.2.1. Процедура read. Процедура *read* имеет следующий вид

```
read (v1, …, vn)
```

Допускается один или больше параметров, каждый из которых является

значением некоторого базисного типа данных. Если *v* является значением типа *char*, то *read(v)* передает следующую литеру входного текста в *v*. Если *v* является значением типа *byte*, *integer, cardinal* или *real*, то *read(v)* означает чтение последовательности литер из входного текста и присваивание прочитанного числа *v*. Предшествующие пробелы и символы конца строки подавляются и отбрасываются.

10.5.2.2. Процедура readln. Процедура *readln* имеет следующий вид

```
readln(v1, …, vn)
```

readln имеет ту же функциональность, что и *read,* за тем исключением, что после чтения *vn* все оставшиеся литеры в строке игнорируются вплоть до (и включая) следующего символа конца строки.

10.5.3. Процедуры вывода

10.5.3.1. Процедура write. Процедура *write* имеет следующий вид

```
write (p1, …, pn)
```

Она может иметь один или несколько параметров, каждый из которых имеет вид

```
e , e:m или e:m:n
```

Здесь *e* изображает значение, которое выводится, а *m* и *n* спецификаторы ширины поля. Если значение *e* требует менее чем *m* литер для своего представления, то пробелы (промежутки) выводятся для того, чтобы гарантировать, что суммарно записано ровно *m* литер. Если *m* опущено, то используется по умолчанию определенное реализацией число. Форма *e:m:n* применима только к числам типа *real* (см. ниже).

Параметры процедуры *write* могут иметь тип *char*, *string*, *boolean*, *integer, cardinal* и *real*.

- Если *e* имеет тип *char*, то *write(e : m)* записывает *m* – 1 пробелов, за которыми следует литера, содержащаяся в *e*. Если *m* опущено, записывается только данная литера.
- Если *e* имеет строковый тип (тип *string*), то *write(e : m)* записывает литеры строки, которым предшествует необходимое число пробе-

лов с тем, чтобы гарантировать суммарный размер поля *m*.

- Если *e* имеет тип *boolean*, то записывается либо слово *true,* либо *false,* которому предшествует необходимое число пробелов с тем, чтобы гарантировать суммарный размер поля *m*.
- Если *e* имеет тип *cardinal* или *integer*, то записывается десятичное представление числа *e,* которому предшествует необходимое число пробелов с тем, чтобы гарантировать суммарный размер поля *m*.
- Если *e* имеет тип *real,* то записывается десятичное представление числа *e,* которому предшествует необходимое число пробелов с тем, чтобы гарантировать суммарный размер поля *m*. Если опущен параметр *n,* то оно имеет вид представления с плавающей точкой, которое состоит из коэффициента и масштабного множителя. Если *n* присутствует, то оно имеет вид представления с фиксированной точкой с *n* знаками после десятичной точки.

10.5.3.2. Процедура writeln. Процедура *writeln* имеет следующий вид:

```
writeln (v1, …, vn)
```

Процедура *writeln* имеет ту же функциональность, что и *write*, с тем отличием, что после записи *vn* помещается символ конца строки.

10.5.3.3. Значения ширины поля по умолчанию для write. Значения параметров ширины поля по умолчанию для *write* и *writeln* зависят от типа параметра следующим образом:

- *char* по умолчанию имеет размер 1,
- *string* по умолчанию имеет размер 4,
- *boolean* по умолчанию имеет размер 6,
- *cardinal* по умолчанию имеет размер 20,
- *integer* по умолчанию имеет размер 20,
- *real* по умолчанию имеет размер 20.

11. Программные единицы

Zonnon-программы могут текстуально разбиваться на единицы, каждая из которых отдельно компилируется. Допускается также текстуальная вложенность некоторых из этих единиц. Правила, управляющие этим, описаны в разд. 2.

11.1. Модуль

Модуль (*module*) имеет двойственную природу: он определяет синтаксический контейнер для логически связанных объявлений и одновременно он определяет объект, чей жизненный цикл управляется системой. Таким образом, модуль обеспечивает механизм для текстуального разбиения исходной программы и также для динамической загрузки во время выполнения части программы в виде экземпляра объекта.

Любое количество динамически созданных объектов могут иметь свои жизненные циклы, управляемые системой, но только единственный экземпляр каждого объекта модуля может быть создан системой в любое заданное время. По этой причине модуль также идеален для реализации абстрактных типов данных.

```
Module = module [ ModuleModifier] ModuleName [ ImplementationClause ] ";"
                [ ImportDeclaration ]
                ModuleDeclarations
                ( BlockStatement | end ) SimpleName.
ModuleModifier = "{" ident "}" // private или public.
ModuleDeclarations = { SimpleDeclaration | NestedUnit ";" }
                { ProcedureDeclaration | OperatorDeclaration }
                { ActivityDeclaration }.
NestedUnit = ( Definition | Implementation | Object ).
ImplementationClause = implements DefinitionName { "," DefinitionName }.
ImportDeclaration = import Import { "," Import } ";".
Import = ImportedName [ as ident ].
ImportedName = ( ModuleName | DefinitionName | ImplementationName | NamespaceName |
ObjectName).
```

Каждый *module* имеет уникальное имя и состоит из текста, который может отдельно быть скомпилирован как единица. Факультативно *module* может реализовывать одно или больше *definition* (см. разд. 2). В этом случае различные аспекты объекта раздельно определяются в единицах *definition*, которые поддерживают абстрактный интерфейс. Факультативно *module* может импортировать элементы из одной или нескольких *implementation*, что означает получение доступа к их областям действия и возможности агрегирования их содержимого. Используя предложение *as*, также возможно переименовать все объекты при их импортировании. Это можно использовать для предотвращения конфликтов имен и/или упрощения длинных внешних имен для повышения программной читаемости.

Пример:

```
import System.Console as S;
...
S.WriteLine('Hello'); (*эквивалентно System.Console.WriteLine('Hello') *)
```

Модуль факультативно может содержать

- другие текстовые единицы, т.е. единицы *definition*, *implementation* и *object,*
- простые объявления констант, типов, переменных и процедур,
- объявления знаков операций для определения определенных пользователем знаков операций,
- объявления активностей для определения активностей в *module* при создании экземпляра.

Примеры:

```
module Small;
begin
      write ('Hello World')
end Small.

module BodyMassIndex;
(* вычислить индекс массы тела *)
      var height, weight, bmi: real;
begin
          write('weight in kg? '); read(weight);
          write('height in m? '); read(height);
          bmi := weight / (height * height);
          write(' body mass index is', bmi : 6: 2);
          if bmi < 19 then
                write('too thin')
          elsif bmi < 27 then
                write('OK')
          else
                write('too fat')
          end
end BodyMassIndex.

definition D; …end D.

definition E; …end E.

module M;
import D, E;
      var a: object{D, E}; (* один объект, который реализует как D, так и E *)
…
end M.
```

Факультативно объект может реализовать один или более *definitions*. В этом случае различные аспекты объекта определяются отдельно в единицах *definition,* которые обеспечивают абстрактный интерфейс. Также объект может импортировать элементы из *module* или *implementation,* т.е. получить доступ к их областям действия. Используя предложение *as,* также возможно переименовать все элементы, когда они импортируются. Это можно использовать для предотвращения конфликтов имен и/или упрощения длинных внешних имен для повышения программной читаемости при программировании внутри объекта.

Заметим, что объект, импортирующий определение *D* для того, чтобы использовать реализацию *D,* должен явно агрегировать ее, импортируя *D* (см. разд. 11.3 и 11.4).

11.2.1. Наследование: уточнение и агрегирование

Имеется два вида наследования *(inheritance)* в языке Zonnon: уточнение (refinement) и агрегирование (aggregation). Уточнение является наследованием некоторого определения интерфейса, в то время как агрегирование является наследованием (переиспользованием) (фрагмента) существующей реализации. Предполагается, что все определения объектов, которые явно не уточняют некоторый другой объект, уточняют *object*. Таким образом, все объекты (прямо или косвенно) уточняют *object*. Если некоторый объект *B* уточняет объект *A*, то говорят, что *B* “происходит из” *A*.

11.2.2. Кратное наследование

Кратное наследование (multiple inheritance) характеризуется возможностью уточнения нескольких определений и/или агрегирования нескольких реализаций. В языке Zonnon отсутствует неоднозначность, связанная с кратным наследованием, поскольку в нем используются квалифицированные идентификаторы для именования (см. разд. 5.1).

11.2.3. Полиморфизм

Полиморфизм (polymorphism) в общем случае включает выбор подходящего метода для вызова во время выполнения в зависимости от типа переменной, которая должна быть задействована. Имеются две концепции:

1) здесь требуется некоторый объект типа *T*, и

2) здесь требуется некоторый объект, который реализует определение интерфейса *D*.

Язык Zonnon делает основной упор на второй более общей концепции, причем идет дальше, разрешая спецификацию кратных определений (так называемые аспекты или грани (*facets*) полного интерфейса объекта) и, таким образом, в этом контексте полиморфизм означает, что «здесь требуется некоторый объект, который реализует *D1* и *D2* и …».

11.3. Определение

Определение (*definition*) определяет другой аспект некоторого объекта в терминах абстрактного интерфейса, состоящего из объявлений полей и сигнатур методов (но не тел методов). Определения могут формировать сеть связанных типов, а не только иерархию. Зависимости между определениями не могут содержать циклы.

```
Definition = definition [ DefinitionModifier] DefinitionName [ RefinementClause ] ";"
        [ ImportDeclaration ]
        DefinitionDeclarations
end SimpleName.
DefinitionModifier = "{" ident "}" // private или public
RefinementClause = refines DefinitionName.
ImportDeclaration = import Import { "," Import } ";".
Import = ImportedName [ as ident ].
ImportedName = ( ModuleName | ImplementationName | NamespaceName |
                DefinitionName| ObjectName).
DefinitionDeclarations = { SimpleDeclaration } { { ProcedureHeading ";" } | ActivitySignature }.
```

Definition имеет уникальное имя и факультативно уточняет (*refines*) другое *definition*, представляя новый аспект объекта, возможно добавляя поля и поведение и, таким образом, формируя специальную форму исходного определения.

Оно может также факультативно импортировать (*import*) элементы из одного или более *implementation*, что дает доступ к их области действия и делает возможным литеральную агрегацию их содержимого. Используя предложение *as,* также возможно переименовать все элементы, когда они импортируются. Это можно использовать для предотвращения конфликтов имен и/или упрощения длинных внешних имен для повышения программной читаемости при программировании внутри объекта. Модификаторы *public* и *private* могут использоваться для объявления видимости содержимого определения. Если модификатор не представлен, то по умолчанию выбирается *public*. *Definition* может содержать множество объявлений констант, типов и переменных, а также заголовков процедур методов (сигнатур), но не тел процедур.

Примеры:

```
definition Graphical;
(* свойства всех графических объектов *)
        var x, y: integer; (* object's position *)
        procedure MoveTo (newX, newY: integer);
        (* post: (x = newX) & (y = newY) *)
        procedure MoveBy (dx, dy: integer);
        procedure Draw;
end Graphical.

definition Rectangle refines Graphical;
(*свойства, специфические для прямоугольных объектов *)
        var width, height: integer;
        procedure Area ( ): integer;
end Rectangle.

implementation Graphical;
(*см. Пример в разделе 11.4 *)
…
end Graphical.
object {ref} Box implements Rectangle;
        procedure Area ( ): integer;
        begin
            return width * height
        end Area;
end Box.
```

11.4. Реализация

Реализация *(implementation)* определяет агрегат фрагментов полей и реализаций методов, предназначенных для переиспользования, при включении в программу посредством одного или нескольких образцов объектов. Реализация имеет уникальное имя, если она не имеет то же имя, что и соответствующее определение (*definition)*. Она факультативно может импортировать (*import*) элементы из одной или нескольких других реализаций (*implementation*s), т.е. получать доступ к их областям действия и иметь возможность агрегировать их содержимое. Используя предложение *as,* также возможно переименовывать все элементы при их импорте. Эта возможность может быть использована для ликвидации конфликтов имен и/или для упрощения длинных внешних имен, что повышает наглядность при программировании внутри объекта. Модификаторы *public* и *private* могут быть использованы для объявления видимости содержимого реализации. Если нет модификатора, по умолчанию выбирается модификатор *public*.

Object, реализующий *definition,* должен реализовывать все его методы, если это определение не имеет соответствующей реализации, которая им-

портируется данным объектом. Однако если объект импортирует *implementation* некоторого *definition* с тем же именем, то предполагается, что он является его (возможно частичной) реализацией.

```
Implementation = implementation [ImplementationModifier] ImplementationName ";"
        [ ImportDeclaration ]
        Declarations
        ( BlockStatement | end ) SimpleName.
ImplementationModifier = "{" ident "}". //private или public
ImportDeclaration = import Import { "," Import } ";".
Import = ImportedName [ as ident ].
ImportedName = ( ModuleName | ImplementationName | NamespaceName |
        DefinitionName | ObjectName ).
```

Implementation может содержать множество объявлений констант, типов и переменных, а также заголовков методов процедур и тел. Эти тела окончательно формируют конкретные реализации методов объектов (т.е. делегаты).

Примеры:

```
implementation Graphical; (* реализация definitionGraphical *)
        (* X и Y объявлены в определении*)
        procedure MoveTo (newX, newY: integer);
        begin
            x := newX; y := newY
        end MoveTo;
        procedure MoveBy (dx, dy: integer);
        begin
            x := x + dx; y := y + dy
        end MoveBy;
end Graphical.
```

12. Отражение

Иногда желательно получить доступ к информации о конструкциях и их атрибутах (например, модификаторах) исходной Zonnon программы. Чтобы сделать это возможным, компилятор может производить *XML*-определение выделенных свойств каждой отдельно транслируемой единицы исходного текста. В дальнейшем к нему может иметь доступ исполняемая программа, используя предопределенную процедуру *getAttribute*. Параметр *construct* является именем любой единицы языка Zonnon, включая программные единицы, типы, константы, переменные, объекты, процедуры, параметры, блоки и знаки операций.

К значениям атрибутов можно получать доступ, используя два вида *getAttribute*:

```
getAttribute(construct, var string);
```

или

```
string := getAttribute (construct);
```

Информация возвращается в виде отдельной строки, возможно содержащей несколько значений атрибутов.

12.1. Схема XML

Следующий список определений XML-схемы используется для описания информации, полученной из программы.

12.1.1. Права доступа

```
<access>public</access>
<access>private</access>
```

12.1.2 Объекты

```
<object>ref</object>
<object>value</object>
```

12.1.3. Параметры процедуры (вид подстановки параметра)

```
<parameter>var</parameter>
<parameter>value</parameter>
```

12.1.4. Неизменяемость процедуры и переменной

```
<immutable>open</immutable>
<immutable>sealed</immutable>
```

12.1.5. Приоритет операции

```
<priority>3</priority>
```

12.1.6. Тела блоков и процедур

```
<behaviour>passive</behaviour> //neither locked nor concurrent
<behaviour>locked</behaviour>
<behaviour>concurrent</behaviour>
```

12.1.7. Ширина типа, переменной и константы

```
<width>64</width>
```

12.1.8. Мощность перечисления

```
<ordinal>7</ordinal>
```

12.2. Пример: программное отражение и информация

```
definition d;
        procedure p1 (var x: integer {32});
        procedure p2 { sealed };
        var v: integer {64};
        type T = ( one, two, three );
end d.

object o implements d;
        procedure p1 (var x: integer {32}) implements d.p1;
            var attrs1, attrs2, attrs3, attrs4, attrs5, attrs6: string;
        begin { locked }
                attrs1 := getAttribute(d);
                attrs2 := getAttribute(d.v);
                attrs3 := getAttribute(p1.x);
                attrs4 := getAttribute(d.T);
                attrs5 := getAttribute(p1);
                attrs6 := getAttribute(d.p2);
        end p1;
        begin
end o.
```

Когда данная программа исполняется, она производит сообщения о своей форме через информацию отражения следующим образом:

```
attrs1(d) contains:
    "<attributes> <access>public</access> </attributes>"

attrs2(d.v) contains:
    "<attributes> <access>public</access> <implement>open</implement>
    <width>64</width> </attributes>"

attrs3(p1.x) contains:
    "<attributes> <parameter>var</parameter> <width>32</width> </attributes>"

attrs4(d.T) contains:
    "<attributes> <access>public</access> <width>32</width> <ordinal>3</ordinal>
    </attributes>"

attrs5(p1) contains:
    "<attributes> <access>public</access> <implement>sealed</implement>
    <behaviour>locked</behaviour> </attributes>"
```

```
attrs6(d.p2) contains:
   "<attributes> <access>public</access> <implement>sealed</implement>
   <behaviour>passive</behaviour> </attributes>"
```

13. Терминология

13.1. Числовые типы

Числовые типы:

- Целые типы — *integer* или *integer*{ширина}
- Кардинальные типы — *cardinal* или *cardinal*{ширина}
- Вещественные типы — *real* или *real*{ширина}

13.2. Идентичные типы

Две переменные *a* и *b* с типами *Ta* и *Tb* являются переменными одного и того же типа, если

- *Ta* и *Tb* оба обозначаются одним и тем же идентификатором, или
- *Ta* объявлен равным *Tb* в объявлении типа вида *Ta = Tb*, или
- *a* и *b* появляются в одном и том же списке идентификаторов в объявлении переменной, поля объекта или формального параметра и не являются открытыми массивами.

13.3. Равные типы

Два типа *Ta* и *Tb* являются равными, если

- *Ta* и *Tb* являются одним и тем же типом, или
- *Ta* и *Tb* являются типами открытых массивов с равными типами элементов, или
- *Ta* и *Tb* являются типами процедур, чьи списки формальных параметров соответствуют.

13.4. Совместимость по присваиванию

Выражение e типа *Te* является совместимым по присваиванию с переменной *v* типа *Tv*, если выполнено одно из следующих свойств:

- *Te* и *Tv* имеют один и тот же тип;
- внутри каждого из типовых семейств *char*, *set*, *integer, cardinal* выражение типа *Te* может быть присвоено переменной *v*, чей тип *Tv* является достаточно большим (определенным посредством своей

ширины) для охвата множества значений типа *Te*;

- *Tv* является типом процедур и *e* является *nil;*
- *Tv* является типом процедур, и *e* является именем процедуры, чей список формальных параметров соответствует списку *Tv.*

13.5. Совместимые как массивы

Фактический параметр типа *Ta* является совместимым как массив с формальным параметром *f* типа *Tf ,* если

- *Tf* и *Ta* являются одним и тем же типом, или
- *Tf* является открытым массивом, *Ta* является массивом, и типы их элементов совместимы как массивы.

13.6. Совместимые в выражении и перегрузка операций

Для данной операции типы операндов считаются совместимыми в выражении, если они соответствуют следующей таблице (которая также показывает тип результата), например: *op1 > op2*. Таблица также неявно демонстрирует комбинации операндов, которые поддерживаются для перегрузки операций.

Операция	*Первый операнд (op1)*	*Второй операнд (op2)*	*Тип результата*
+ - *	integer{m}	integer{n}	max of integer{m} and integer{n}
+ - *	cardinal{m}	cardinal{n}	max of cardinal{m} and cardinal{n}
+ - *	real{m}	real{n}	max of real{m} and real{n}
/	real{m}	real{n} предусловие: op2 # 0	max of real{m} and real{n}
+ - *	set{m}	set{n}	max of set{m} and set{n}
div mod	integer{m}	integer{n} предусловие: op2 # 0	max of integer{m} and integer{n}
or & ~	boolean	boolean	boolean
= # < <= > >=	integer{m}	integer{n}	boolean
= # < <= > >=	cardinal{m}	cardinal{n}	boolean
= # < <= > >=	real{m}	real{n}	boolean
= # < <= > >=	enumeration T	enumeration T	boolean
= # < <= > >=	char	char	boolean
= # < <= > >=	character array,	character array	boolean
= # < <= > >=	string	string	boolean
= #	boolean	boolean	boolean
= #	set	set	boolean
= #	procedure type T	procedure type T	boolean
= #	**nil**	**nil**	boolean
in	integer	set	boolean
implements	object	definition	boolean
is	object	object type	boolean

13.7. Согласованные списки формальных параметров

Два списка формальных параметров согласуются, если справедливы следующие свойства:

- они имеют одно и то же число параметров;
- они имеют либо один и тот же тип результата функции, либо его нет;
- параметры на соответствующих позициях имеют равный тип;
- пары параметров на соответствующих позициях являются одновременно либо параметрами переменными, либо значениями.

ОГЛАВЛЕНИЕ

Елена Викторовна Касьянова

АДАПТИВНЫЕ МЕТОДЫ И СРЕДСТВА ПОДДЕРЖКИ ДИСТАНЦИОННОГО ОБУЧЕНИЯ ПРОГРАММИРОВАНИЮ

Под редакцией
проф. Виктора Николаевича Касьянова

Рукопись поступила в редакцию 28.05.2007
Редактор З. В. Скок

Подписано в печать 16.07.07
Формат бумаги 60 × 84 1/16 Объем 9.7 уч.-изд.л., 10.6 п.л.
Тираж 60 экз.

Центр оперативной печати «Оригинал 2»
г. Бердск, ул. Островского, 55, оф. 02, тел. (383) 214-45-35